Roadiz Documentation

Release 2.2.0

Ambroise Maupate, Julien Blanchet

Dec 12, 2023

Contents

1 Philosophy **3**

2 User documentation **5**
2.1 User documentation . . . 5
2.1.1 Write in Markdown . . . 6
2.1.2 Managing nodes . . . 8
2.1.3 Managing node-types . . . 19
2.1.4 Managing documents . . . 24
2.1.5 Managing users . . . 26

3 Developer documentation **33**
3.1 Developer documentation . . . 33
3.1.1 First steps . . . 33
3.1.2 Node system . . . 43
3.1.3 Building headless websites using API . . . 54
3.1.4 Tag system . . . 66
3.1.5 Documents system . . . 68
3.1.6 Attributes . . . 72
3.1.7 Forms . . . 75
3.1.8 Contributing . . . 85
3.1.9 Troubleshooting . . . 86

4 Extension documentation **87**
4.1 Extension system . . . 87
4.1.1 Extending Roadiz . . . 87
4.1.2 Events . . . 89
4.1.3 Extending Solr indexation . . . 91

Index **95**

Roadiz is a polymorphic CMS based on a node system that can handle many types of services. It is based on *Symfony* framework, *Doctrine ORM*, *API Platform* and *Twig* for maximum performances and security.

Roadiz node system allows you to create your data schema and to organize your content as you want. We designed it to break technical constraints when you create tailor-made websites architectures and layouts.

Imagine you need to display your graphic design portfolio and... sell some t-shirts. With Roadiz you will be able to create your content forms from scratch and choose the right fields you need. Images and texts for your projects. Images, texts, prices and even geolocation for your products. That's why it's called *polymorphic*.

Note: This is documentation for Roadiz v2.x, browse v1.x documentation

CHAPTER 1

Philosophy

When discovering Roadiz back-office interface, you will notice that there aren't any Rich text editor also called *WYSIWYG* editors. We chose to promote *Markdown* syntax in order to focus on content hierarchy and quality instead of content style. Our guideline is to preserve and respect the web-designers and graphic designers work.

You'll see that we built Roadiz as web-designers and for web-designers. It will allow you to create really quickly website prototypes using *Twig* templates or develop complex headless websites with the power of *API Platform*.

We want Roadiz to be a great tool for designers and developers to build strong web experiences together. But we thought of editors too! Roadiz back-office theme "Rozier" has been designed to offer every back-users a great writing and administrating experience.

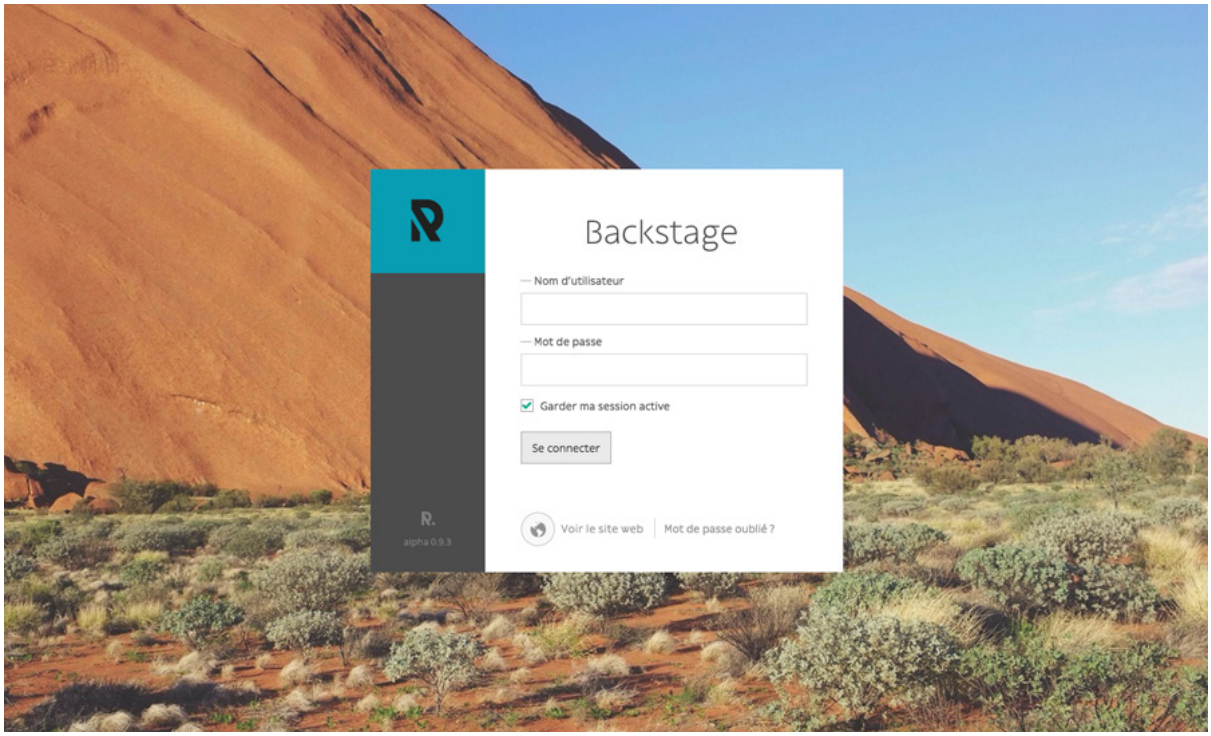
CHAPTER 2

User documentation

2.1 User documentation

Note: User documentation is on the way. We invite you to send us questions on our Gitter account or to leave some documentation suggestions on our Github repository.

First of all, you will need to connect to Roadiz' backoffice in order to make changes to your website contents. To connect, you just have to write `/rz-admin` after your website domain name, in your browser address bar. Then you will be able to enter your username and password that you chose during Roadiz installation or that you received by email.



Here you can choose to keep your connection active for a couple of days, if your browser accepts cookies.

If you forgot your credentials, the *Forgot password?* section will ask you an email to send you a password reset link.

Table of contents

2.1.1 Write in Markdown

> Markdown is a lightweight markup language with plain text formatting syntax designed so that it can be converted to HTML and many other formats using a tool by the same name. [...] The key design goal is readability – that the language be readable as-is, without looking like it has been marked up with tags or formatting instructions, unlike text formatted with a markup language, such as Rich Text Format (RTF) or HTML, which have obvious tags and formatting instructions.
>
> —Wikipedia article — https://en.wikipedia.org/wiki/Markdown

Titles

Add two hashtag # or more according to your title importance level. Backoffice shortcut buttons allow to directly insert your titles marks before your selected text. Make sure to leave a blank line before each new title you write.

```
## Architecture
### Modern architecture
```

Be careful not to use only one hashtag to create a first-level title as this is usually used for pages main title.

Alternate syntax

Main title and second level titles can be written using = and − as underline characters.

```
Architecture
============

Modern architecture
-------------------
```

Bold

Insert two stars * before and after your text to set in bold. Backoffice shortcut button allows to insert directly the 4 characters around your selected text.

```
This is a **bold text.** And a normal one.
```

Be careful not to leave whitespaces inside your stars group (in the same way you do with parenthesis) otherwise your text won't be styled.

Italic

Insert one star * before and after your text to set in italic. Backoffice shortcut button allows to insert directly the 2 characters around your selected text.

```
This is an *italic text.* And a normal one.
```

Bold and italic marks can of course be combined using 3 stars before and after your selected text.

What if * character is already in use

Bold and italic markup can be performed using _ (underscore) character too if you actually need to write a *star* character in your text.

```
A __3* Bed & Breakfast__ has just opened its doors in middletown.
```

Strike-through

Insert two tildes ~ before and after your text to strike-through.

```
This is ~~striked-through text~~.
```

Ordered and unordered lists

Insert a star * or a dash - followed by a single whitespace for each of your list item. One item per line. Leave a blank line before and after your list. For *ordered* list, use a digit followed by a dot and a whitespace instead.

```
* A line
- An other line
* A unknown line

1. The first item
2. The second item
3. The third item
```

If you need to break an item into several lines, you'll need to use the line-break markup.

Nested list

You can insert a second/third/... level for your list, just **by leaving four spaces before your new list-item mark.**

```
- A list item
    - A sub-item
    - A second sub-item
        1. An ordered sub-sub-item
        2. The second sub-sub-item
```

New paragraph and line-break

A simple line-break is always ignored by Markdown language because it makes a difference between a **paragraph** and a **line-break**. To simply create a line-break without creating a new paragraph, **leave at least 3 spaces at the end of your text line then go to a new line.**

```
Address:<space><space><space>
26 rue Burdeau<space><space><space>
69001 Lyon<space><space><space>
France
```

To create a new paragraph, **always leave a blank line between your text blocks.** Any additional blank line will be ignored.

```
Nullam quis risus eget urna mollis ornare vel eu leo.
Cras justo odio, dapibus ac facilisis in, egestas eget quam.

Aenean eu leo quam. Pellentesque ornare sem lacinia
quam venenatis vestibulum.
```

According to your website design (CSS), new paragraphs may have no visual margins between them. Inserting more than one blank line won't add any additional visual space as Markdown ignores it.

Hypertext links

Write link label between braces immediately followed by the URL between parenthesis. For external links do not forget protocol prefix `http://` or `https://`.

```
[My link](http://www.google.com)
```

To create a internal link, just use relative notation:

```
[Contact us](/page/contact-us)
```

Then, for an email link, use `mailto:` prefix:

```
[John Doe](mailto:jdoe@example.com)
```

A link title can be added by inserting it before *ending parenthesis* wrapped in quotes.

```
[My link](http://www.google.com "Link to Google website")
```

Block quotes

Insert a > **sign before each new paragraph and a space** to wrap your text in a quote block. You can then use all other Markdown symbols inside your quote.

```
> ### Donec ullamcorper nulla non metus auctor fringilla.
> Aenean lacinia **bibendum** nulla sed consectetur.
> Vestibulum id ligula porta felis euismod semper.
```

Images

Images use the link syntax with an exclamation mark prefix `!`. For external images do not forget to write full URL with protocol `http://` or `https://`.

```
![A cat](/files/cat.jpg)

![A cat from an other website](https://www.example.com/images/cat.jpg)
```

Be careful, images will be displayed as is, unless your webdesigner planned to adapt image size coming from Markdown fields using CSS. As links, an external image may break if its owner deletes the original image. Make sure to host critical images directly on your website and to use *relative URL*.

Footnotes

Footnotes are not supported with *basic* Markdown syntax, but the *Markdown Extra* one. So before using them, **make sure your webdesigner used the right Markdown parser in your theme.**

```
Praesent commodo cursus magna[^note], Sed posuere consectetur est at
lobortis. Vel scelerisque nisl consectetur et[^othernote].

[^note]: This a footnote
[^othernote]: This a second footnote
```

Markdown will *automatically generate anchor links between your footnote and its reference*. It will automatically **use numbers as footnote reference labels**, so you don't have to bother to write numbers yourself but easy-to-remember markers labels.

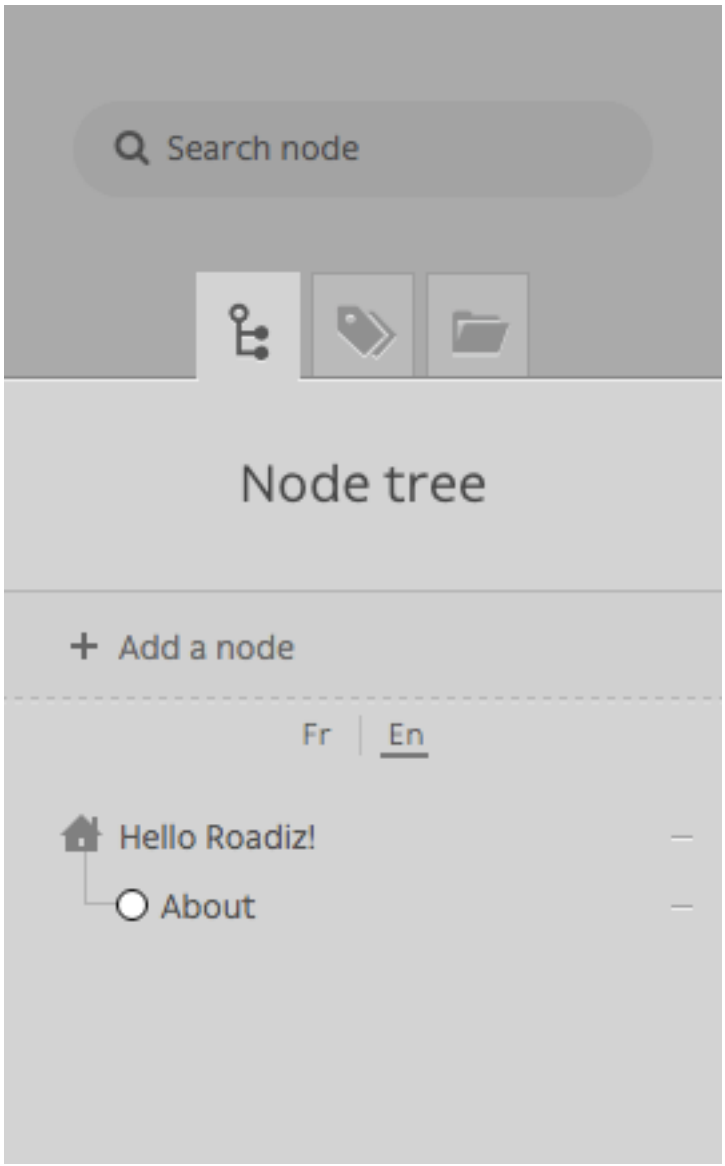
2.1.2 Managing nodes

Nodes are the most important part of Roadiz CMS. They are your content which can be shaped as you want according to your node-types definitions. A node can be a blog-post, a content page, a photo gallery, even a shop product. This is why we called it "Node", it's an abstract data container interconnected with other node in a tree.

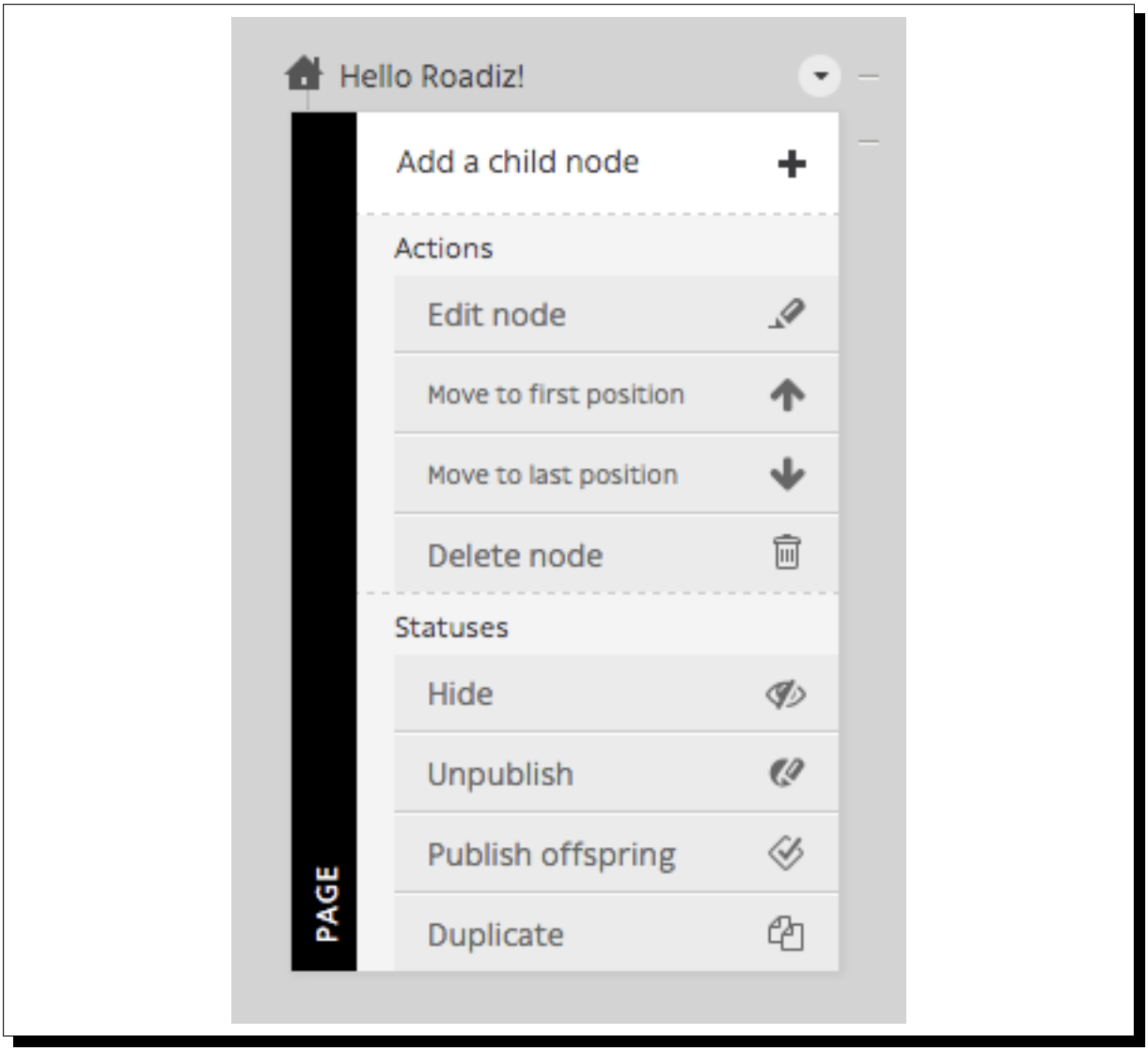
Node-tree navigation

Node tree

Simple node-tree after installing Roadiz with *Default theme*.



Contextual menu for *home* node.



Each node has a place in your website, that's why we chose to arrange your content in a tree-shaped way. It's intuitive and it works the same as your computer files.

- To edit a node' content, simply click on it.
- To move a node across your tree, drag & drop using its handle (round or rombus shape). You can drop a node after or before an other one. You can also drop inside just by moving your mouse a bit on the right, you should see the node shadow to shift right.
- Other actions are available from each node contextual menu. Right click on the node or click on the arrow at the right when you pass your mouse over.

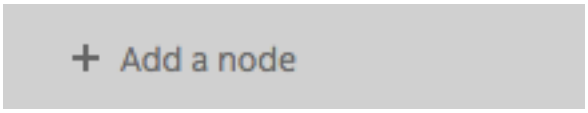
Contextual menu actions

- *Add child node:* to create a content inside the current node.
- *Edit node:* links to the current node "edit content" page.
- *Move to first position:* to move a node at the first position inside its parent node.
- *Move to last position:* basically the same for the last position.
- *Delete node:* to move current node to the trashcan. A confirmation page will be prompt before really deleting a node.
- *Hide/Show:* Change a node' visibility. A hidden node won't be displayed in Urls and your website, even if you are an administrator.

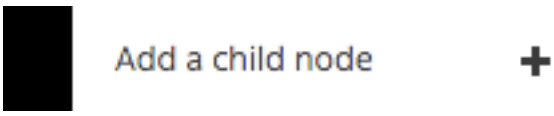
- *Publish/Unpublish:* Change a node' publication status. Unpublished nodes are not visible to anonymous visitors, but visible for back-office users using `?_preview=1` query parameter.
- *Publish offspring:* Publish a node and all its children nodes recursively.
- *Duplicate:* Copy all current node' content and relationships into a new node.

Creating a node

Add buttons



"Add node" button located at the top of your *node-tree*.



"Add a *child* node" button, which is located at the top of each node' contextual menu.

To add a blank node to your node-tree, you will need to choose your location. In Roadiz you can add a content at the root of your tree or choose a "parent-node". In both cases you will be asked to choose a *node-type* and a *node-name* before creating your node.

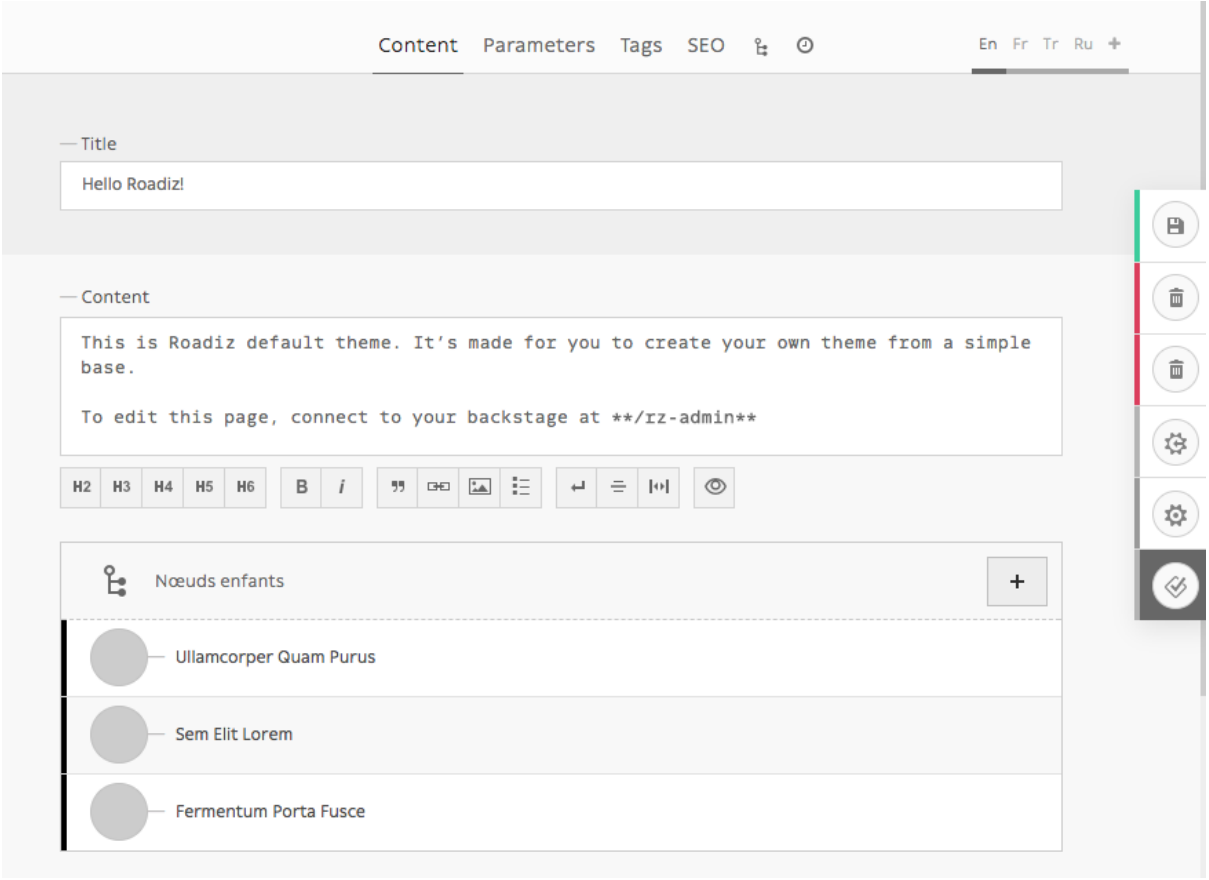
- *Node name* is the global identifier for your node. It must be unique and won't change from one translation to an other. It can be changed later except if your developer locked it up. *Node name* is usually used to build your pages URL.
- *Node-type* defines what fields will be available in your content. Choose well as you won't be able to change it later, you 'll have to delete and recreate an other node.

Edit an existing node

Node edition page is composed in several tabs.

- *Node content*
- *Node parameters*
- *Tags*
- *SEO*
- *Tree*, if your node is set up as a *stack*

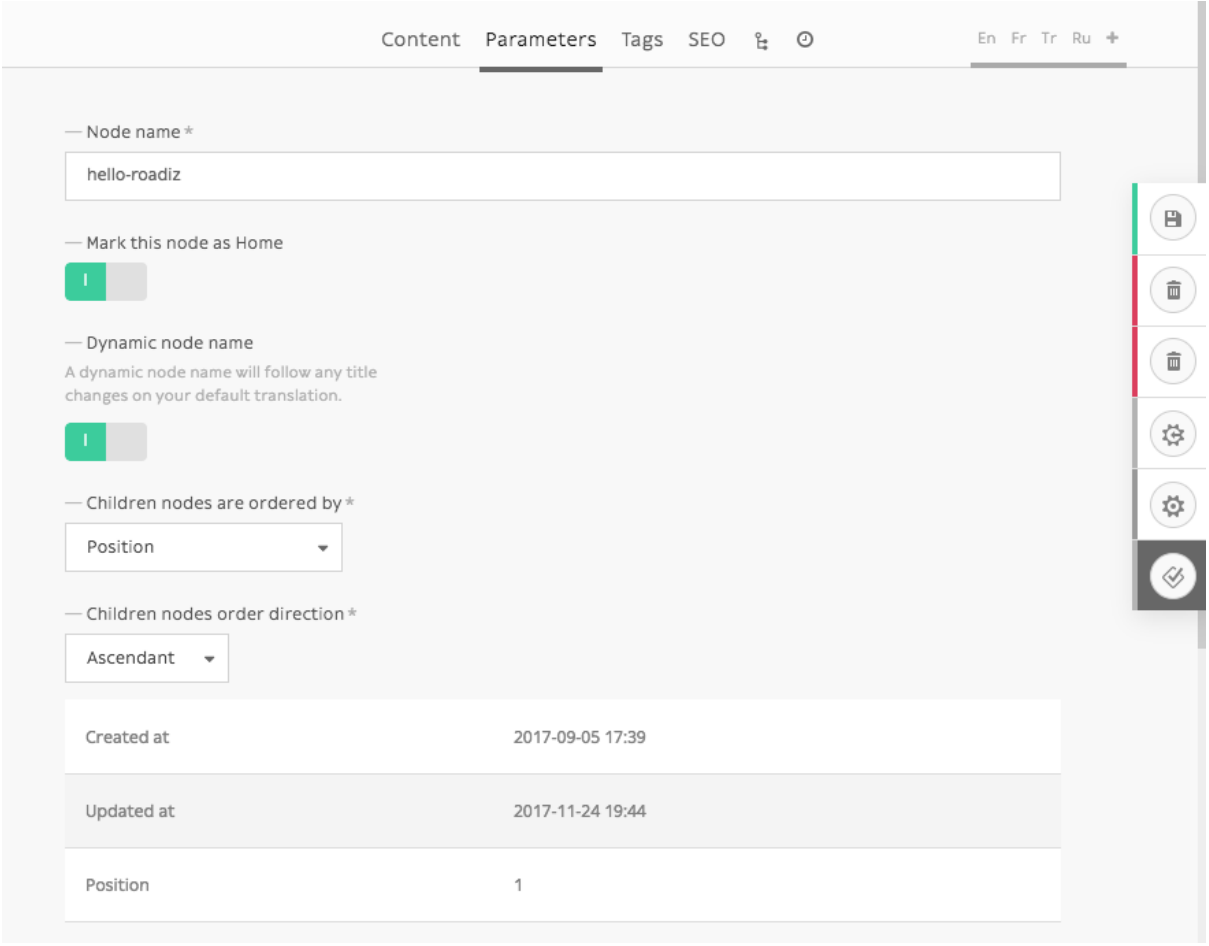
Node content



Contents tab is basically the main part where you will edit your node specific data, using node-type fields such as *text* fields, or *documents* fields, etc.

This tab will display different content over translations depending on which fields are marked as *universal* or not.

Node parameters

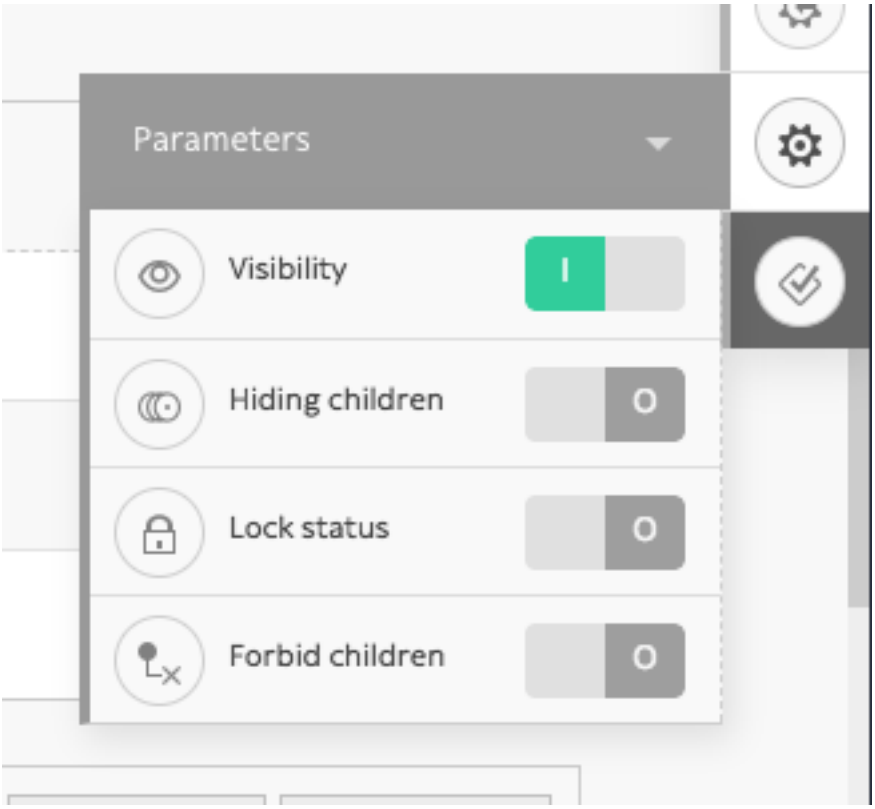


Parameters are global data such as your *node name*. They are used for managing your node visibility according to each user role and node back-office' settings. This section should not be used so often as parameters will be set once by your developer.

This tab will display the same content over translations.

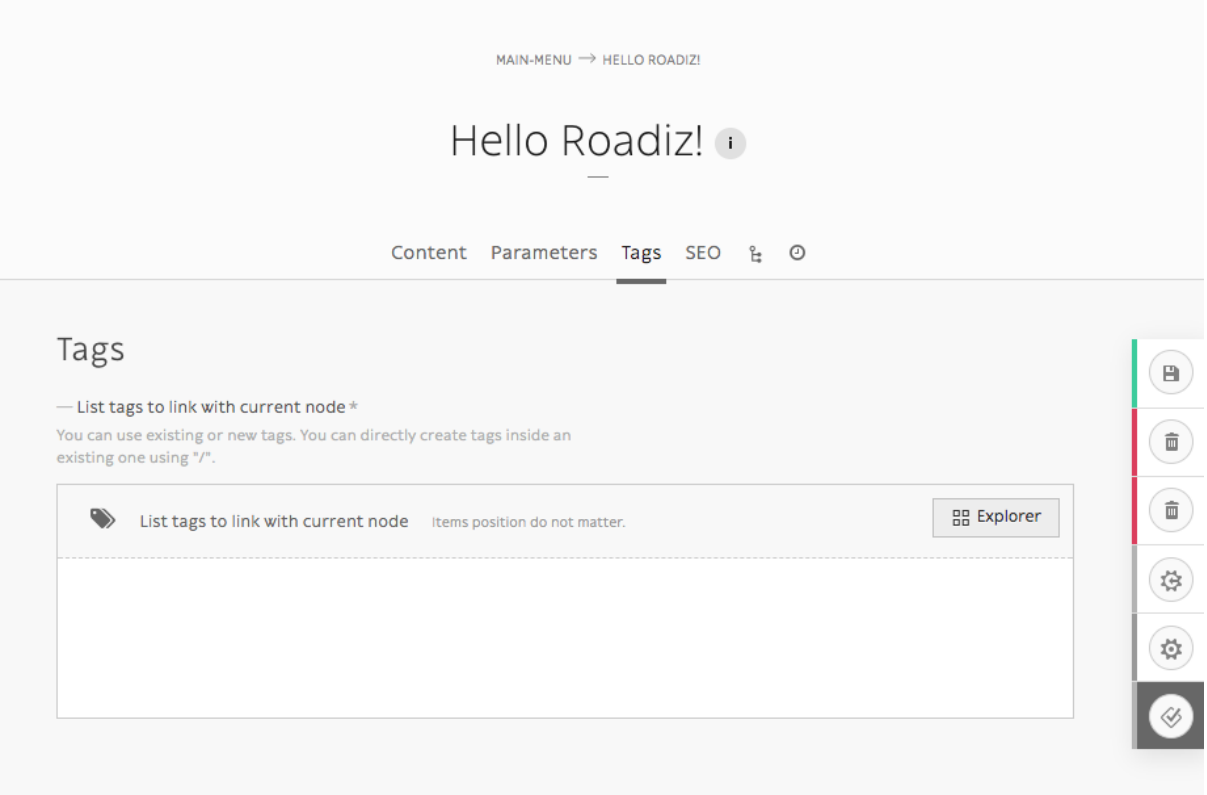
Side actions and parameters

Additional parameters are available on the right action-menu. These parameters are node-wide, they will apply on each node translations.



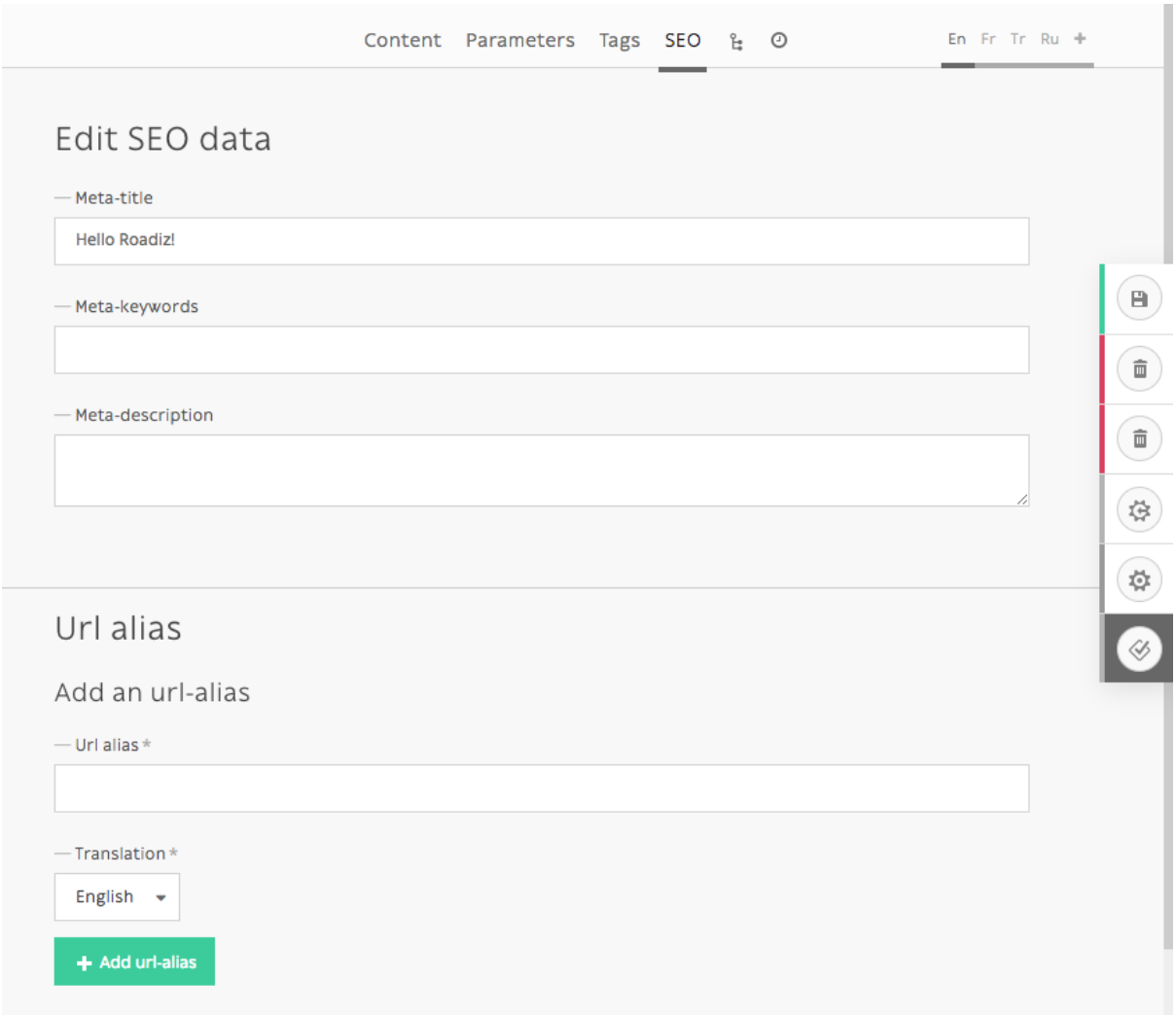
- **Visibility**: Hide or show current node (according to your theme)
- **Hide children**: Switch current node to a **stack**, children nodes won't appear in the global Node-Tree anymore. You will be able to add *Stack type* buttons in the *node parameters tab*.
- **Lock status**: Prevent users to delete current node or rename it. You should switch it on if your rely on some nodes in your themes.
- **Forbid children**: Prevent users to create children nodes.

Tags



This tab will display the same content over translations.

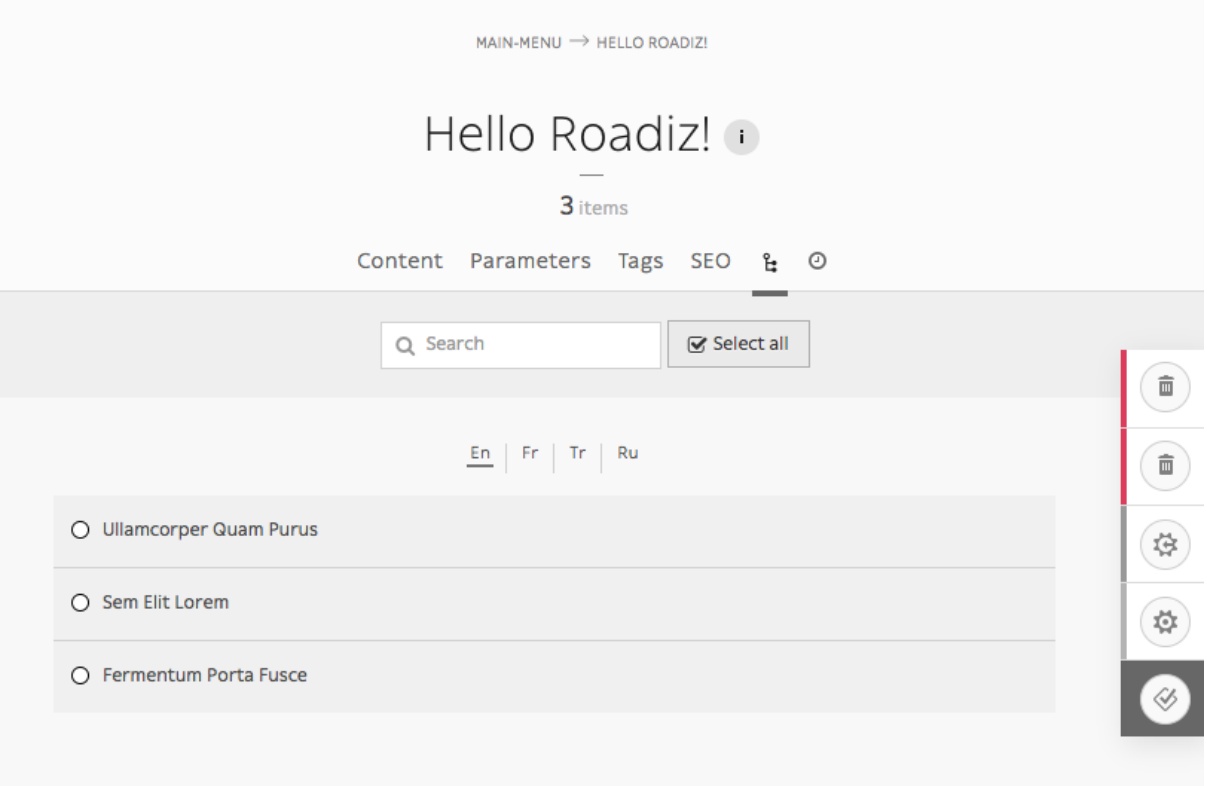
SEO



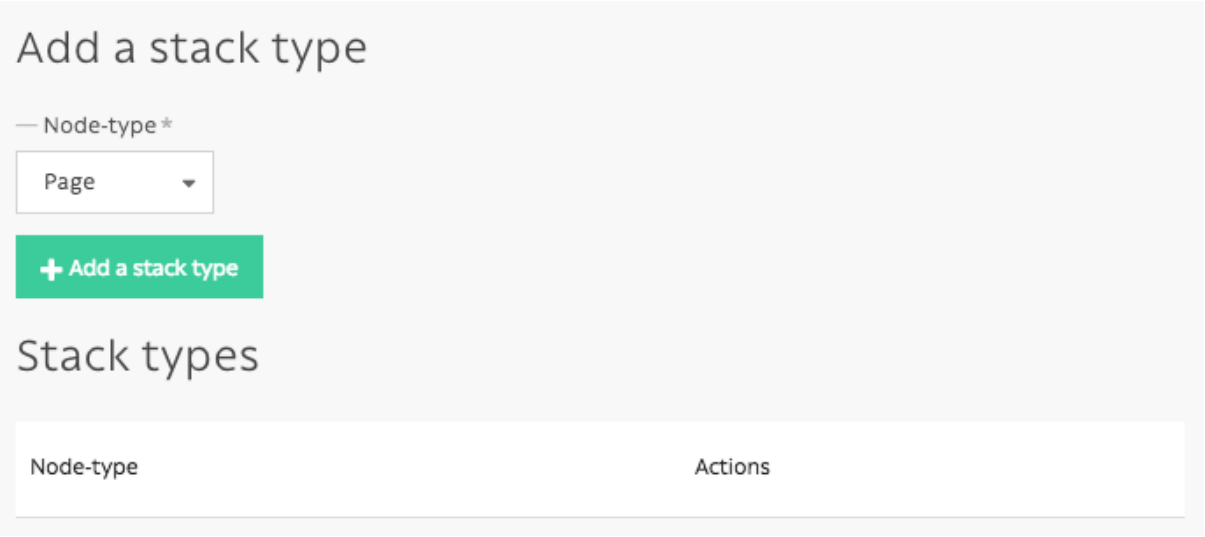
This tab will display different content over translations.

Tree

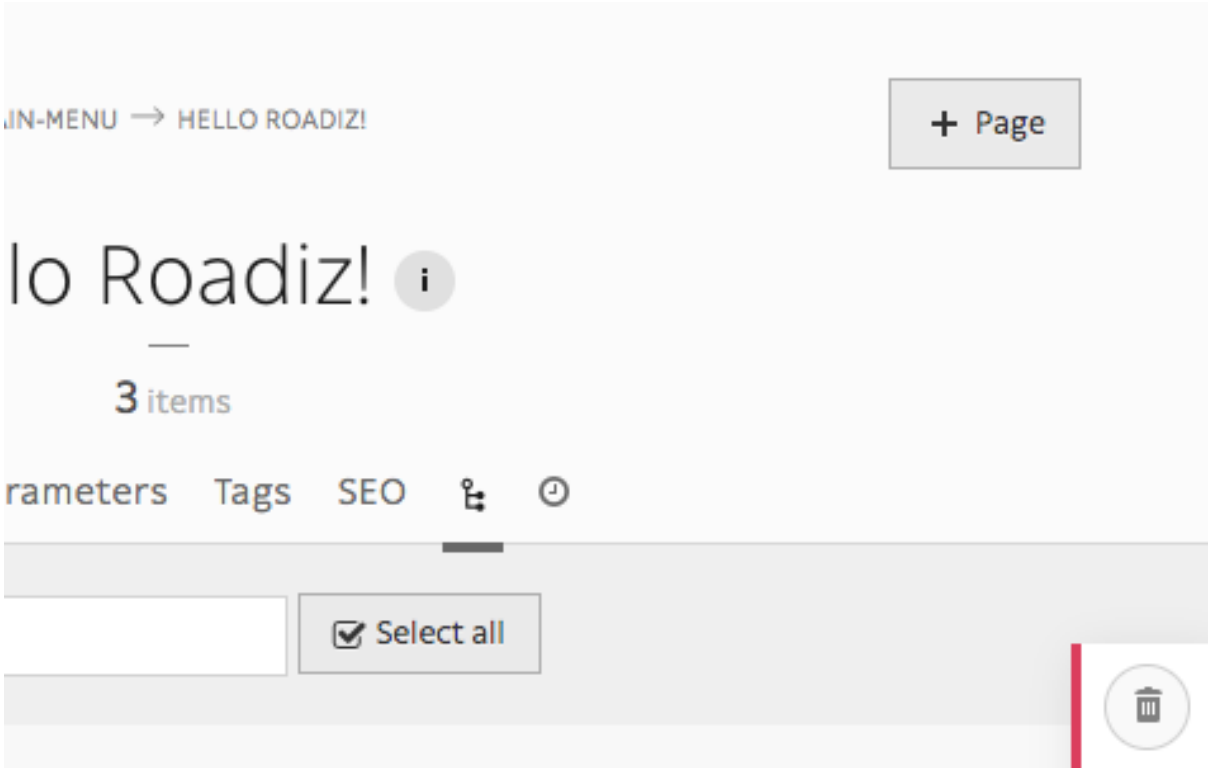
When a node is defined as a *stack*, *Tree* becomes its default view. You can set the default ordering in the *Edit tab*. And node-types set as *publishable* will display their publication date before its name.



Tree view becomes very interesting if you have many many children nodes, when you have blog posts or portfolio projects for example.



Then you can add *stack type* in *Edit tab* to make some handy quick-add buttons to this view.



Nodes publication system

During its lifecycle, every nodes can have a different publication status. When you create a new content, it will be automatically set as **Draft** by Roadiz so that you can edit it without bothering your visitors and sharing unfinished work.

Available statuses:

- **Draft**: First status for new nodes
- **Pending validation**: It's a medium status for user that do not have permission to publish nodes
- **Published**: That's the most important status, it will set the green light to your visitor to view your content
- **Archived**: When you don't want to publish a node but you don't want to delete it either
- **Deleted**: It's the last status for your nodes. Before emptying your node trashcan, every content will wait with this status.

To improve status visibility, *draft* and *pending* nodes have a rhombus shape and *published* nodes have a circle shape.

Preview unpublished nodes

Unpublished nodes are not viewable for anonymous visitors, Roadiz allows backend users to preview them adding a dedicated *query parameter* called `?_preview=1` in your website URLs. Using a different URL than your public website is very important as it could lead to errors or to expose unpublished content if your website is hosted behind a reverse proxy cache like *Varnish*.

For example, if your `my-news` page is not published yet, connecting to `http://mywebsite.com/my-news` will lead to a 404 page for your anonymous visitors, as well as you too. If you want to preview it, you'll have to connect to `http://mywebsite.com/my-news?_preview=1`. This URL will only allow authenticated backend users, other people will be denied.

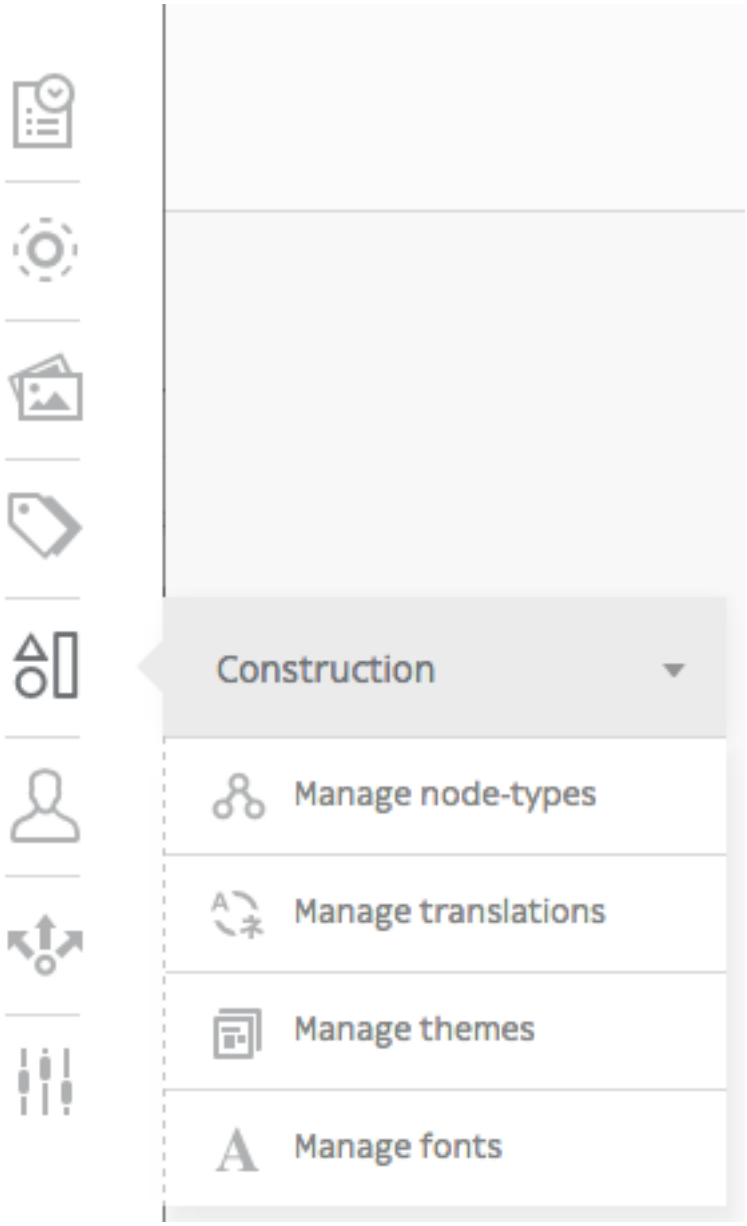
2.1.3 Managing node-types

This is a simple guide on how to create and manage nodes using Roadiz CLI, add and remove node fields, or even how to import nodes.

First and foremost, you need to create a new node-type before creating any kind of node.

If you want to know more about what a node-type is, please visit the other section of the developer documentation.

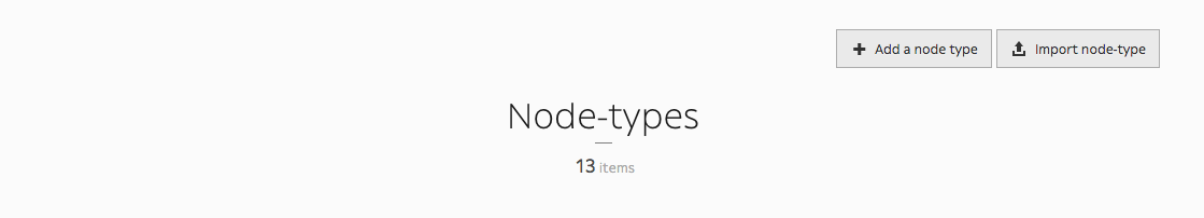
When working with Roadiz in the back-office, you can easily manage node-types via the *Construction* icon in the toolbar.



Add node-type

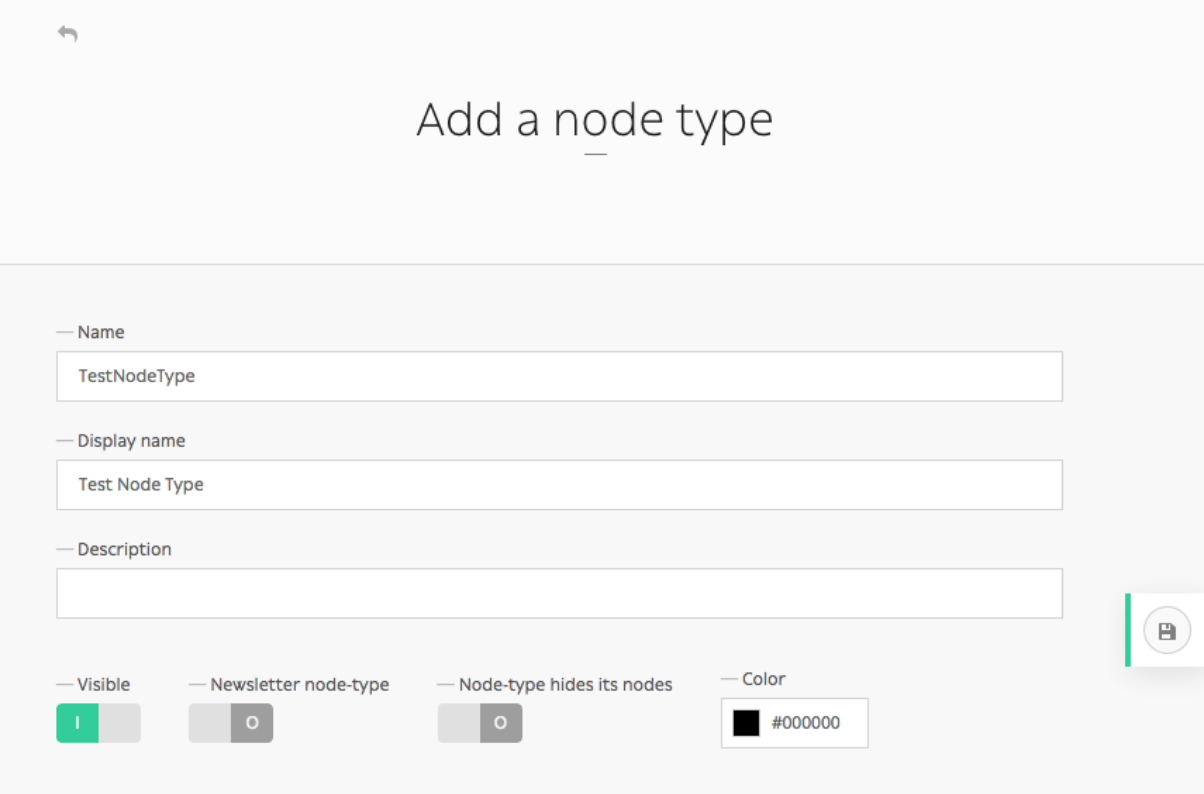
Once you have landed on the Node-Types page (*https://mywebsite.com/rz-admin/node-types*), you can create node-types by clicking on *Add a node-type*.

Note: You can export and import a node-type if you have a .json file. See *Other action* for more information.



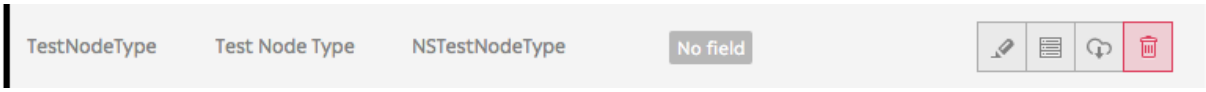
Upon filling the two mandatory settings *Name* (that developers will use) and *Display Name* (that back-office users will see), you are now ready to create your first node type.

Warning: Be careful when you name your node-type though, *Name* field can't be changed once the node-type is created. See *Delete node-type* section to know how to delete a node-type.



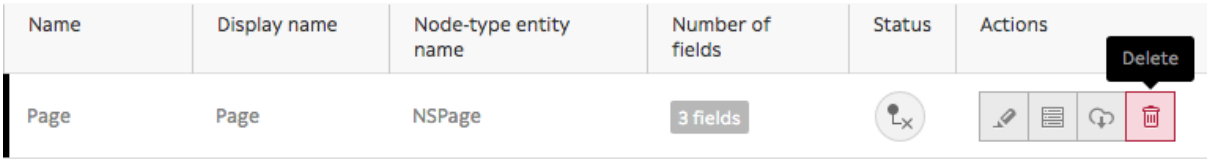
Other options (*Visible*, *Newsletter node-type*, *Node-type hides its nodes* and customizable color) aren't required and can always be altered later on.

You have now created your first node-type! It now appears on the node-type page along other node-types and you can now manage its fields.



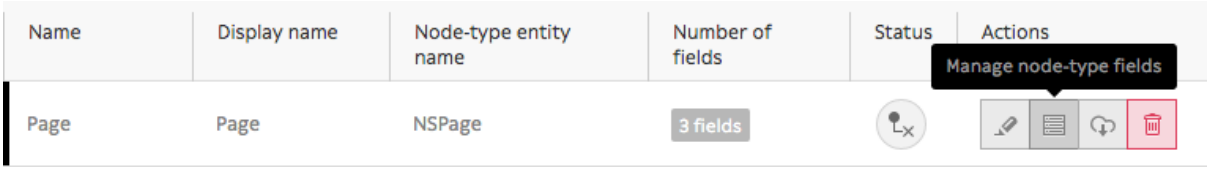
Delete node-type

Made a typo when creating a node-type? No longer in need of a node-type ? You can delete it by simply clicking the trashcan icon on the Node Types page (*https://mywebsite.com/rz-admin/node-types*).



Adding node-type field

To add fields to your newly-created node-type, click the *Manage node-type fields* icon.



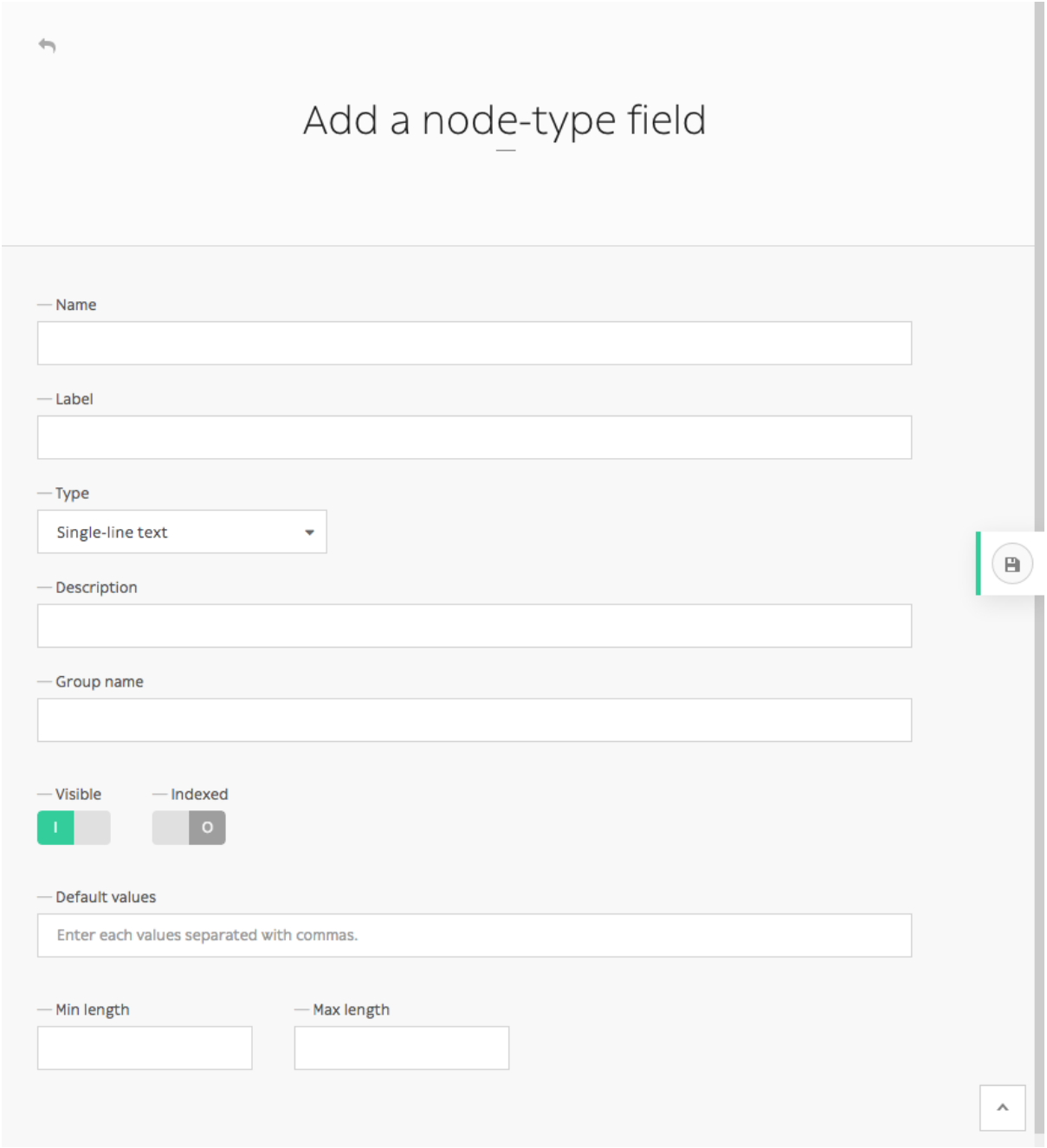
Then click on 'Add a node-type field' button.



Fill in the form to create a new field :

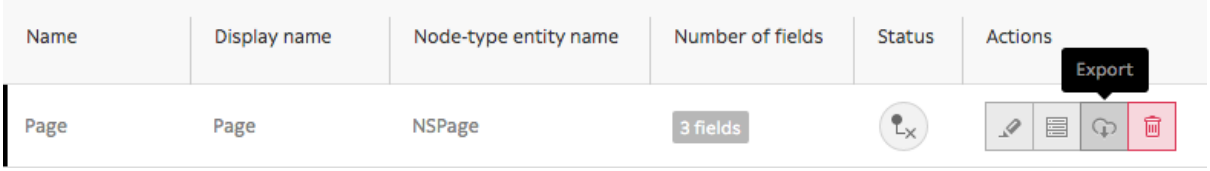
- *Name*: what developers will use
- *Label*: what back-office users will see
- *Type*: single choice option that will define the content (basic text, markdown text, documents, email, number, single or multiple choice, children nodes etc.)
- *Description*, *Group name*, *Visible*, *Indexed*, *Default values*, *Min length*, *Max length* (optional)

Note: *Default values* is an interesting field as you can specify what kind of node-types that can be linked to this node-type. You can also use it as a filter in the explorer, and only show those default values.



Other actions

From one website to another, you can export node-types as .json files.



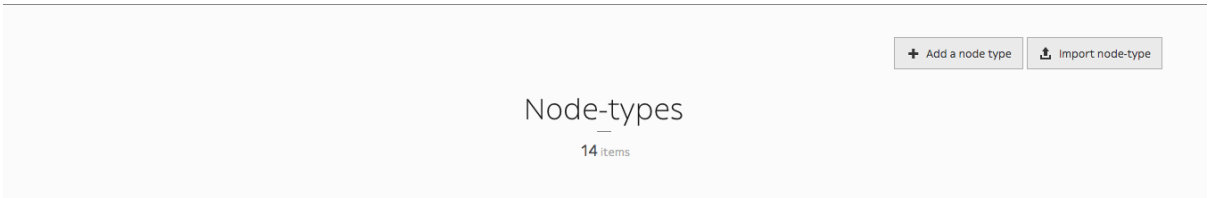
An .json file should look like this when you open it in any editor:

```
{
    "name": "Page",
    "displayName": "Page",
    "visible": true,
    "publishable": false,
```

```
    "reachable": true,
    "newsletterType": false,
    "hidingNodes": false,
    "color": "#000000",
    "fields": [
        {
            "position": 1.0,
            "name": "content",
            "label": "Content",
            "type": 4,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 2.0,
            "name": "children",
            "label": "N\u0153uds enfants",
            "type": 16,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 3.0,
            "name": "header_image",
            "label": "Header image",
            "type": 9,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 4.0,
            "name": "pictures",
            "label": "Pictures",
            "type": 9,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        }
    ],
    "defaultTtl": 0
}
```


Notice the four fields that have been added to this *Page* node-type.

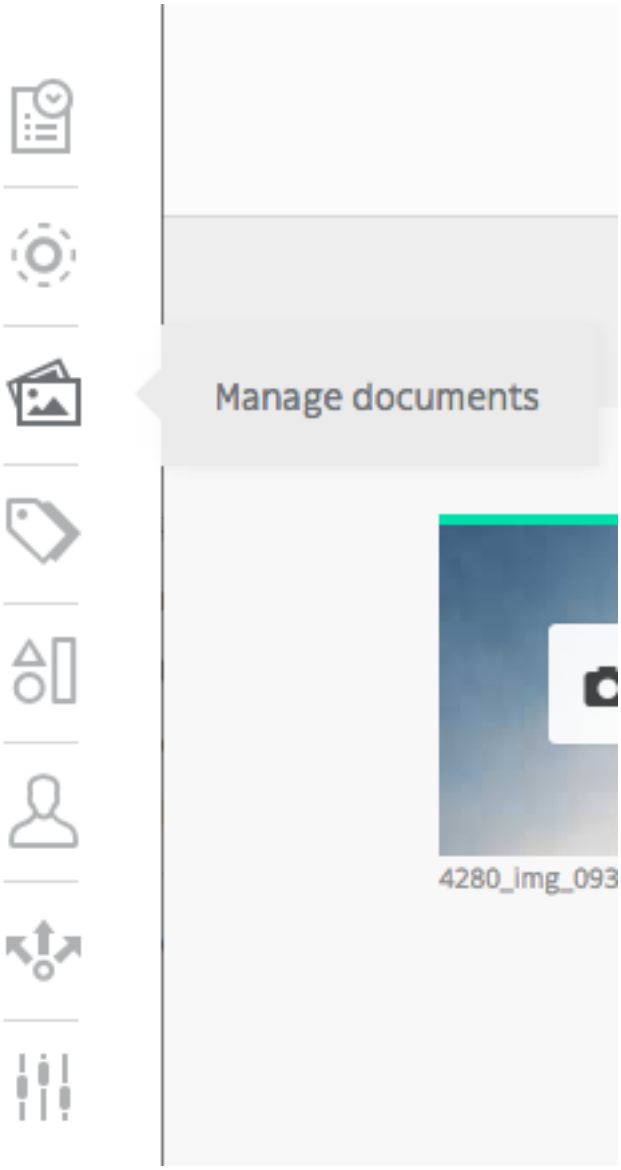
You can write an .json file yourself if you feel like it, but it is probably easier to simply export node-types from existing website, then import it on your new website. It can be easily done by clicking on *Import node-type* button, then selecting it via your explorer.



You are close to fully master Roadiz back-office powers! Keep referring to this documentation if you have any problem, and let us know if any information goes missing.

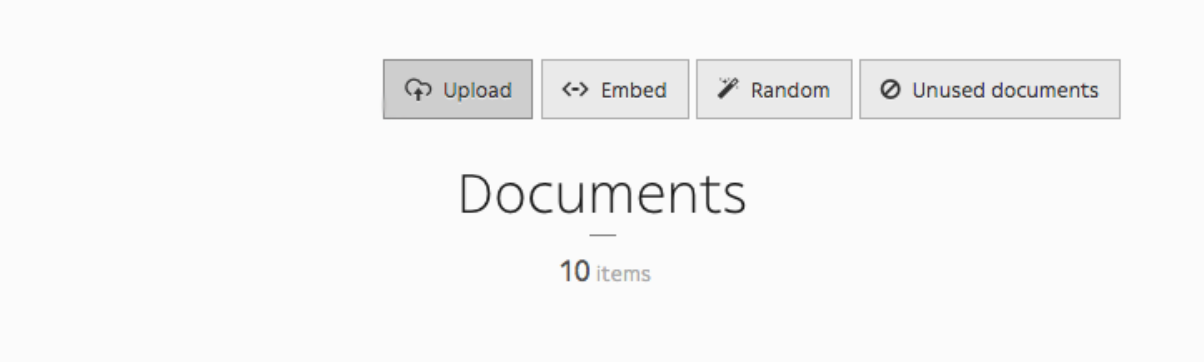
2.1.4 Managing documents

You can manage documents via the *Document* icon in the toolbar. You can either upload, embed, randomly downloaded documents, and even have a look at unused documents in your website.



Upload document

Uploading a document has never been this easy: just drag and drop the document in the designated area.



When you upload files, Roadiz will try to reduce any image larger than 2500 pixels (width or height) in order to ease up further transformations on it. The CMS will always keep a *raw* version of your uploaded document if you decide to increase this limit over 2500 pixels. Keep in mind that Roadiz won't be able to optimize your image to a better format: for example if you uploaded a PNG image containing no transparent area, the CMS won't convert it to JPEG format and you will end up with a heavy file even if you resize it.

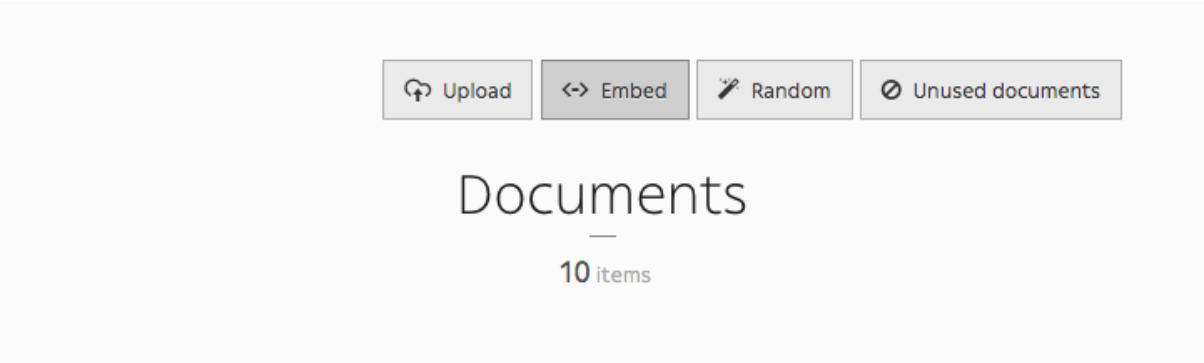
- Photographs, complex images with no transparency: *use JPEG*
- Logos, icons, complex images with transparency: *use PNG*

Using an image optimizer (such as *tinyjpg*, *kraken.io*, *jpgoptim*, ...) **before** uploading onto Roadiz is useless because, the CMS will lose these optimizations when it processes transformations (crop, resizing, contrast and quality).

On the other hand, if your website has been developed with `picture` support, Roadiz will be able to serve any images in the new WEBP format which is 30% lighter than JPEG and supports transparency (alpha channel). So any user browsing your website with a Chrome, Firefox or Edge will see only WEBP images and will save up minimum 30% of bandwidth usage. In the near future, AVIF image format will be more supported by major browsers, and Roadiz will be able to generate optimized image with this format too.

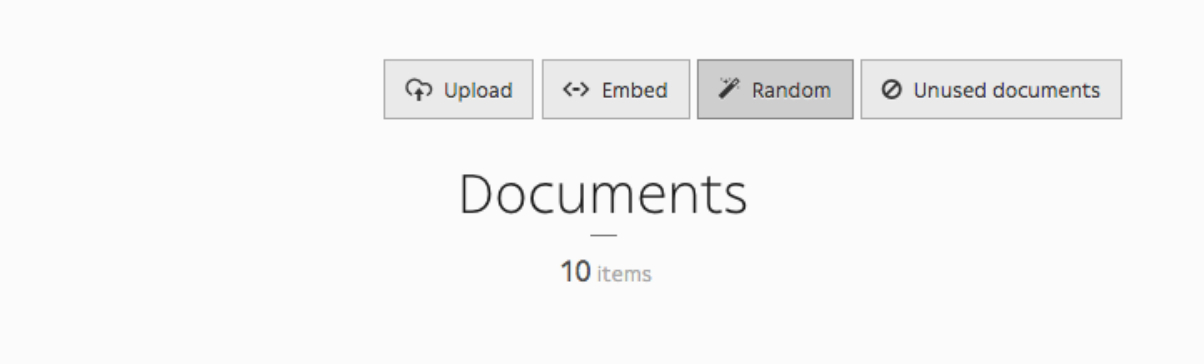
Embed document

Here is the magical part of the Documents section. You can have embedded documents from *YouTube*, *SoundCloud*, *Vimeo*, *Mixcloud*, *Spotify*, *Twitch*, *TED* and *Dailymotion*, then use them as if they were images in your content blocks.



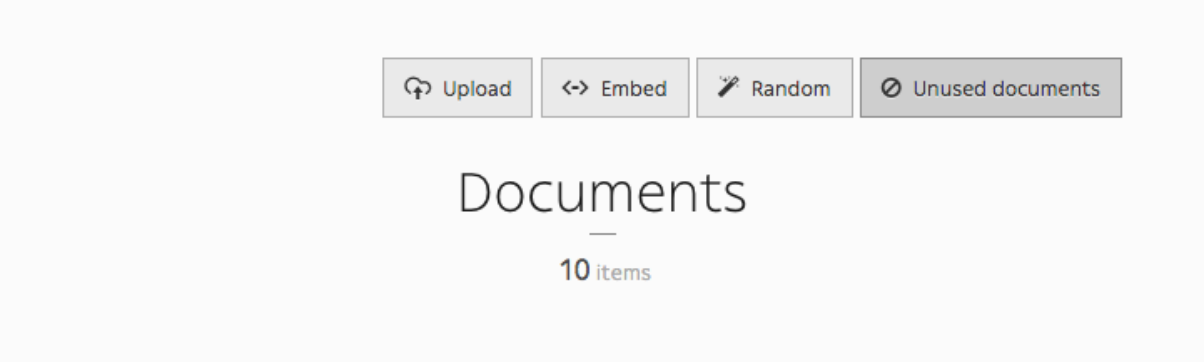
Random document

Random is a cool feature that allows to download random documents to your website from *Splashbase*.



Unused document

Clicking *Unused document* allows you to gather every unused documents on your website, so you can clean your database and get rid of useless documents.

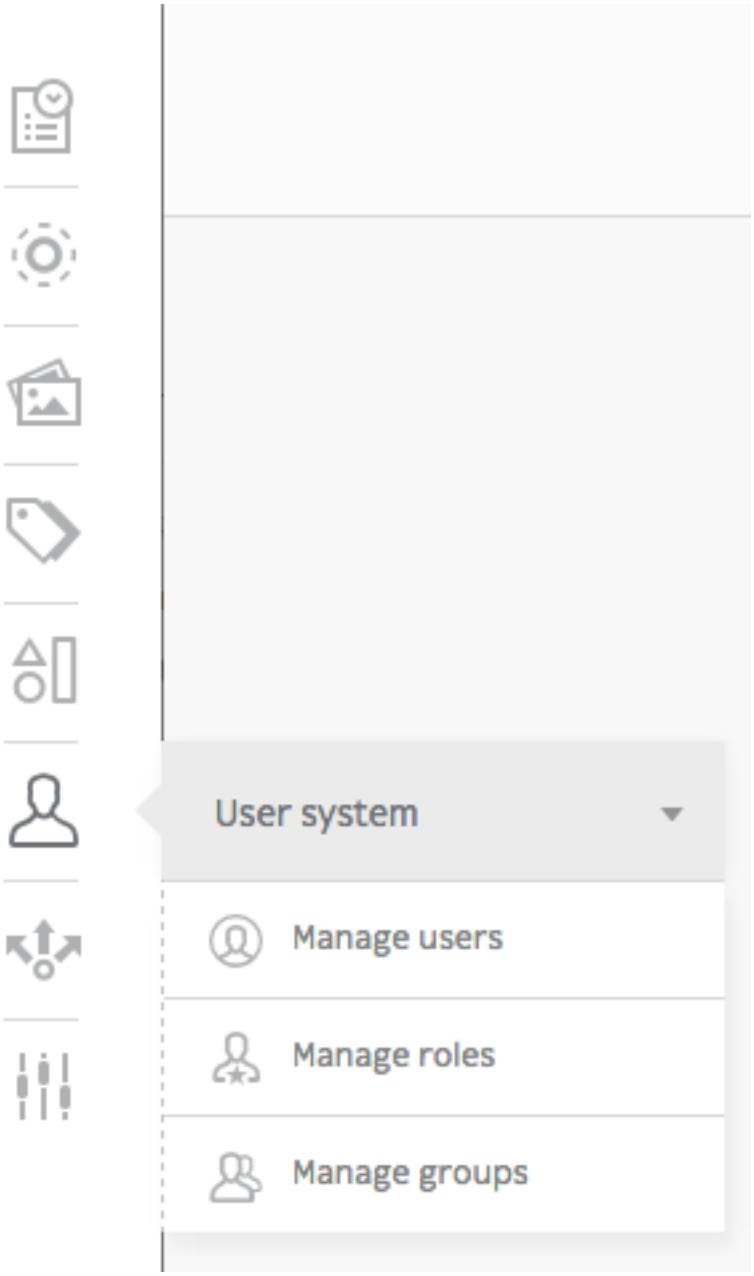


2.1.5 Managing users

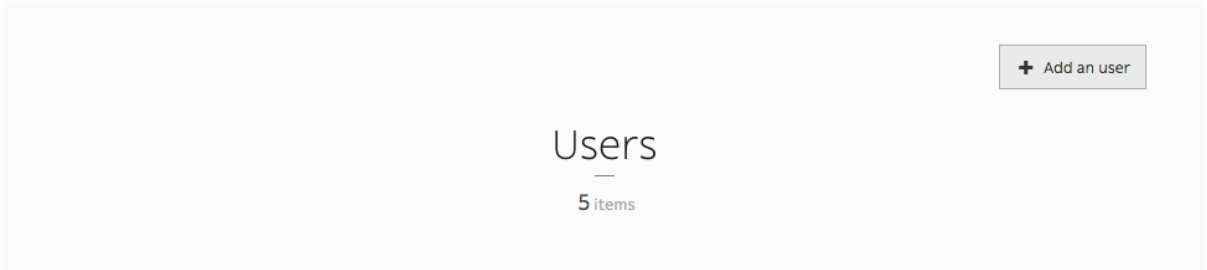
This is a simple guide on how to create and manage users using Roadiz CLI.

There are two ways of adding users, via the back-office and in command-line, both will be displayed in each section.

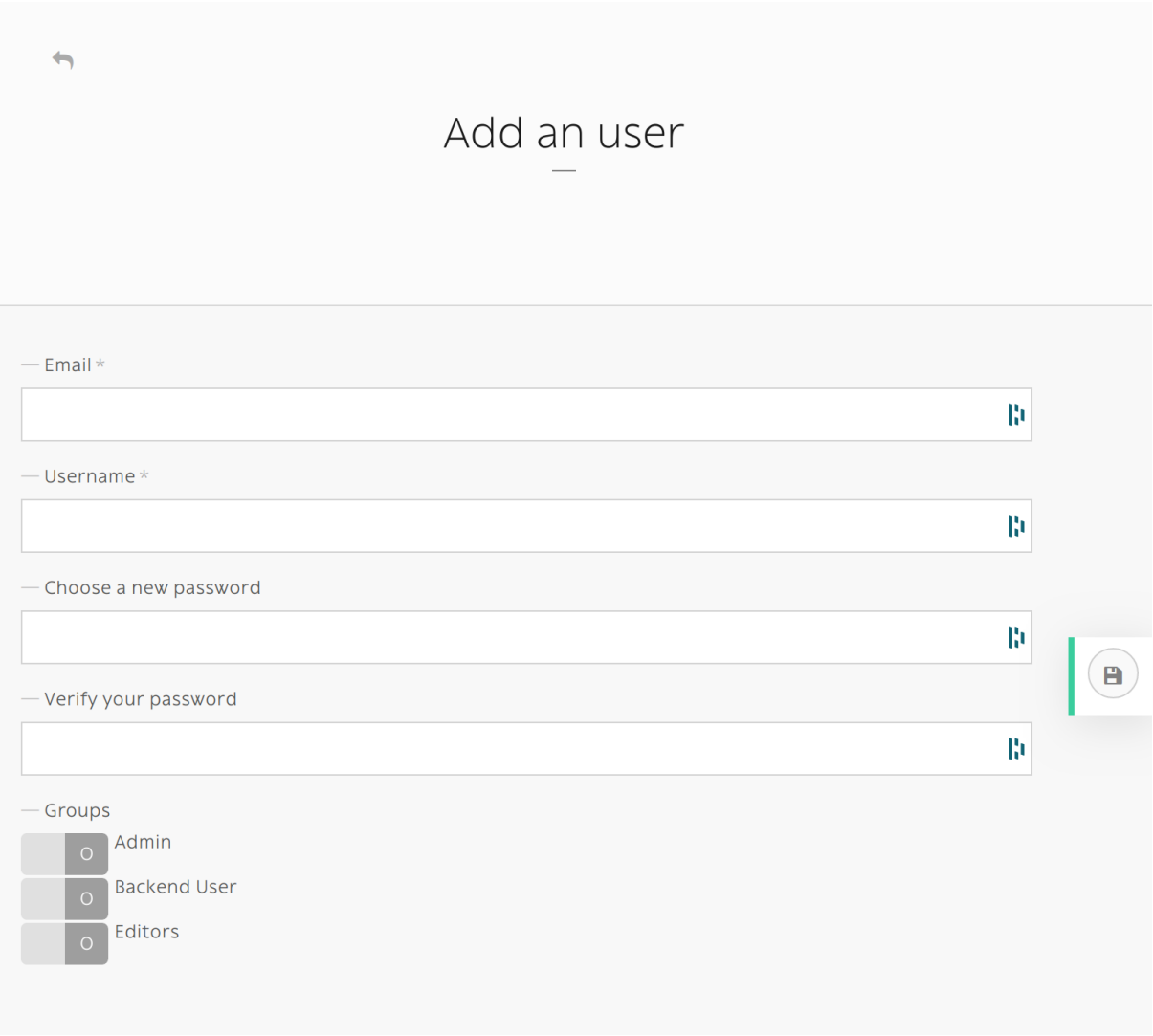
When working with Roadiz in the back-office, you can easily manage users via the *User system* icon in the toolbar.



Create a new user



You can add users simply by clicking *Add an user* button.



If you create an user without specifying its *password*, an email with a password reset link will sent. Make sure that you entered the right email and that your Roadiz website has a *mailer* configured. New user will be locked unless he or she resets its password first.

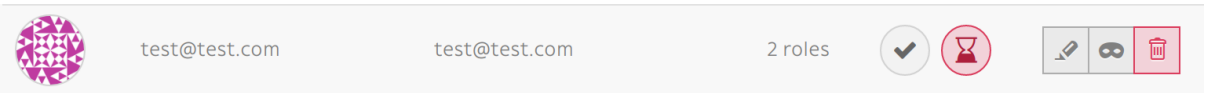
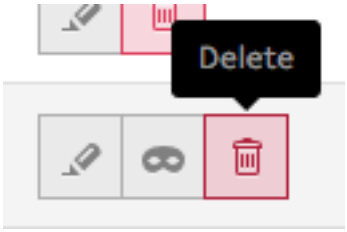


Fig. 2.1: A new user without password will stay locked until he or she resets its password.

The command-line `bin/console users:create loginname` starts a new interactive user creation session. You will create a new user with login and email, you can also choose if it's a backend user and if it's a superadmin.

Delete user

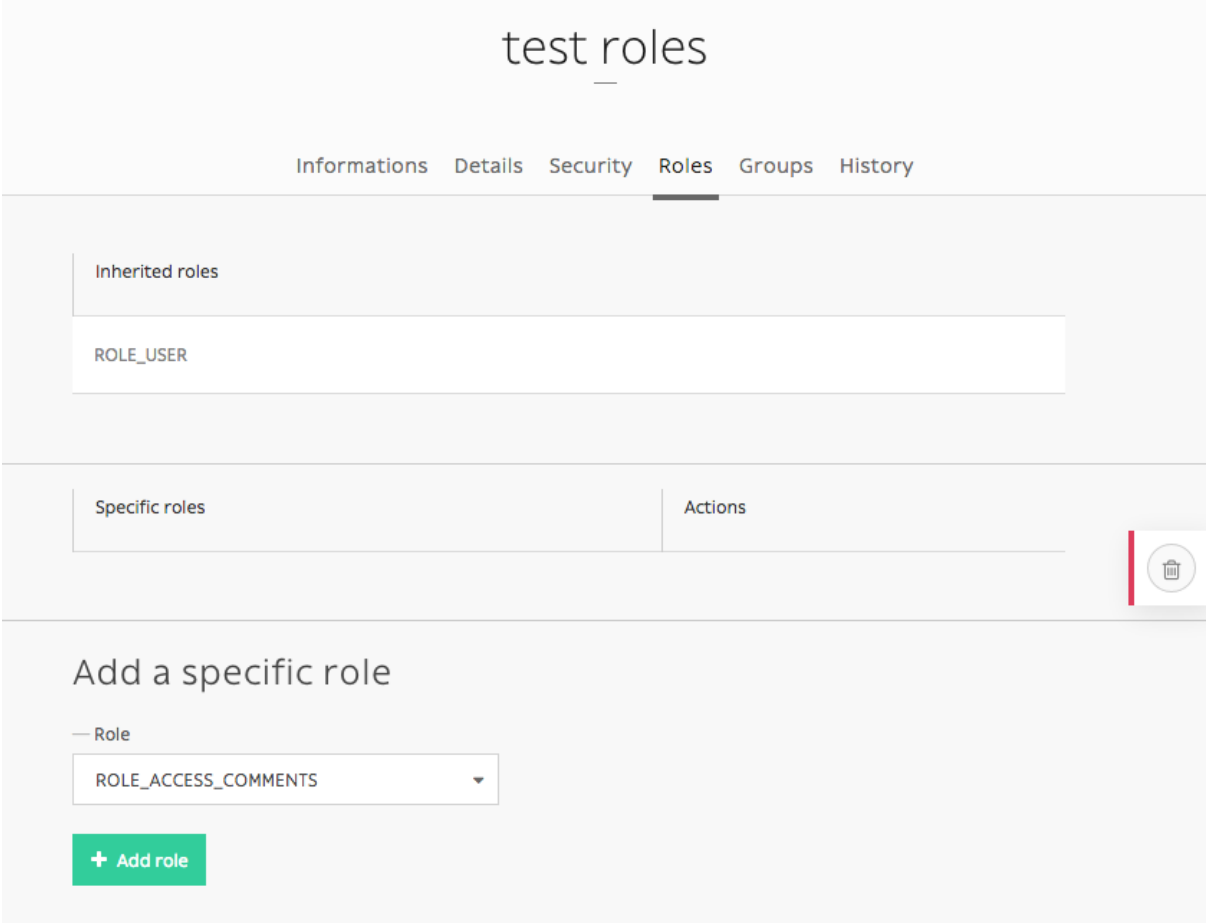
You can remove users by clicking the trashcan icon.



The command `bin/console users:delete loginname` delete the user "loginname".

Adding role

You can edit a users profile the same way you edit a node-type. You can add roles in the *Roles* tab.



If you want to add `ROLE_SUPERADMIN` role to "test" user, it would look like this in command-line:

```
bin/console users:roles --add loginname
# You will be prompted to choose the ROLE with auto-complete feature.
```

Other action

It is possible to enable or disable users with `users:enable` or `users:disable` command. If a user doesn't remember his password, you can regenerate it with the `users:password` command. For more information and more actions, we invite you to check available commands with:

```
bin/console list users
```

Enable SSO for back-office users

Roadiz is compatible with any **OpenID** single sign-on system (such as Google, Exchange, ...) and can be configured to allow your company users to login to your back-office with several `ROLES`.

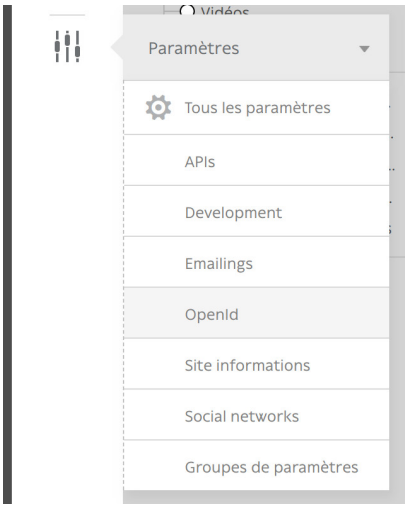
First, make sure to generate and gather the following information from your *OpenID* provider:

- OpenID client ID
- OpenID client secret
- OpenID auto-discovery URL (i.e. for Google GSuite user https://accounts.google.com/.well-known/openid-configuration)

Then you should decide:

- What *roles* (comma separated) you want to be automatically granted to users connected with your SSO.
- What domain name to restrict users from. This is very important for *Google Suite* users because the auto-discovery is the same for everybody. You may not want to allow every Google Suite users to access your back-office .
- A button label for your back-office login page.

```
# config/packages/roadiz_rozier.yaml
roadiz_rozier:
    open_id:
        # Verify User info in JWT at each login
        verify_user_info: false
        # Standard OpenID autodiscovery URL, required to enable OpenId login in␣
↪Roadiz CMS.
        discovery_url: '%env(string:OPEN_ID_DISCOVERY_URL)%'
        # For public identity providers (such as Google), restrict users emails by␣
↪their domain.
        hosted_domain: '%env(string:OPEN_ID_HOSTED_DOMAIN)%'
        # OpenID identity provider OAuth2 client ID
        oauth_client_id: '%env(string:OPEN_ID_CLIENT_ID)%'
        # OpenID identity provider OAuth2 client secret
        oauth_client_secret: '%env(string:OPEN_ID_CLIENT_SECRET)%'
        granted_roles:
            - ROLE_USER
            - ROLE_BACKEND_USER
            - ROLE_SUPERADMIN
```



Fill all your gathered information to the right *Roadiz* dotenv variables.

```
OPEN_ID_DISCOVERY_URL=https://accounts.google.com/.well-known/openid-configuration
OPEN_ID_HOSTED_DOMAIN=my-google-workspace-domain.com
OPEN_ID_CLIENT_ID=xxxxxxx
OPEN_ID_CLIENT_SECRET=xxxxxxx
```

Then, if your parameter are correct you should see your SSO connect button on *Roadiz* back-office login page. Pay attention that if your SSO users do not have sufficient permissions they may have a 403 error after being redirected from SSO flow.



Note: Google Suite OpenID implementation is described at https://developers.google.com/identity/protocols/oauth2/openid-connect#discovery

First, create a new OAuth2 application on https://console.cloud.google.com/ and follow instructions at https://developers.google.com/identity/protocols/oauth2/openid-connect#getcredentials

CHAPTER 3

Developer documentation

3.1 Developer documentation

3.1.1 First steps

Requirements

Roadiz is a Symfony application running with PHP. You can follow regular Symfony requirements to optimize your local or production setup. Roadiz requires an HTTP server for static assets and **SSH access** with out/ingoing allowed connections. Here is a short summary of mandatory elements before installing Roadiz:

- Nginx or Apache, with a dedicated virtual host as described below.
- PHP 8.1+ **required**, 8.2 recommended
- Install theses PHP extensions (which are installed and enabled by default in most PHP installations): JSON, Intl, cURL, MBString, Ctype, iconv, PCRE, Session, Zip, SimpleXML, and Tokenizer;
- Your php.ini needs to have the `date.timezone` setting
- You need to have at least version 2.6.21 of libxml
- PHP *OPcache*
- `php.ini` recommended settings
  - `short_open_tag = Off`
  - `magic_quotes_gpc = Off`
  - `register_globals = Off`
  - `session.auto_start = Off`
- MariaDB 10.5.2+ or MySQL 8.0+ database with *JSON_** functions support
- Install Composer, which is used to install PHP packages.
- Git

Roadiz is a Symfony application

Roadiz is not meant to be deployed directly on a production server right out-of-the-box, it is a Symfony application that you must configure and customize on your development environment then commit your own project repository

configuration, migrations. Then you will be able to deploy it using your preferred method (SFTP / SSH / Git / Docker). Remember that as you would do with any Symfony app, you'll have to clear cache, run migrations and other stuff each time you deploy to a new environment. This may require a SSH access to your production env or building a Docker image with a custom entrypoint script.

Using Docker as a development and production environment

Roadiz and **Symfony** development and production environments heavily rely on Docker and docker-compose because it eases up development and deployments stages using tools such as *Gitlab* or *Github Actions*. We recommend creating Docker images containing **all your project sources and dependencies**.

You can use our official Docker images with *PHP-FPM* and *Nginx* already setup for you. We recommend that you create your own Docker image based on this official one.

- https://hub.docker.com/r/roadiz/php82-fpm-alpine PHP-FPM 8.2 container ready for Roadiz and Symfony apps. Used for main application, async workers and cron jobs.
- https://hub.docker.com/r/roadiz/nginx-alpine Nginx container ready for Roadiz and Symfony apps. Used for static assets and proxying to PHP-FPM container.

`docker-compose` is meant to be used on the host machine (especially on Windows and macOs hosts). *Docker* is not mandatory if you prefer to install PHP and a web server directly on your host, just follow official Symfony instructions : https://symfony.com/doc/current/setup.html#technical-requirements

One container per process

Since Roadiz v2.1, we recommend separating processes into different docker containers. This allows you to scale each process independently. For example, you can have multiple PHP-FPM containers running your application, but only one Nginx container serving static assets. You can also have multiple PHP-FPM containers running your application, but only one Redis container for your cache. This allows you to scale each process independently.

Create a new Roadiz project

For new projects **Roadiz** can be easily setup using `create-project` command and our *Skeleton*.

```
# Create a new Roadiz project
composer create-project roadiz/skeleton my-website
cd my-website
# Create a local Dotenv to store your secrets
cp .env .env.local
# Edit your docker-compose parameter in .env to
# fit your development environment (OS, UID).
# .env file will be tracked by Git
#
# Initialize your Docker environment
docker-compose build
docker-compose up -d --force-recreate
```

Warning: **Roadiz** and **Symfony** development and production environments heavily rely on Docker and docker-compose. We recommend you to learn these awesome tools if you're not using them yet. You still can use Roadiz without Docker, but you will have to install and configure a *PHP* environment, *MySQL* database, and a web server. If you are not using *docker* or *docker-compose*, just ignore `docker-compose exec -u www-data app` prefix in the following commands.

Note: Keep in mind that Roadiz v2 is a complete rewrite to become a true *Symfony* Bundle, it is true a *Symfony* app and behaves like that. Roadiz v2 is meant to be used as a headless CMS with *API Platform*. But you still can use *Controllers* and *Twig* templates, but there is no more theme logic, just Symfony Bundles and your own code (in `./src` folder).

Composer will prompt you if you want to can versioning history. Choose the default answer `no` as we definitely want to replace *roadiz/skeleton Git* with our own versioning. Then you will be able to customize every files in your projects and save them using Git, not only your theme. Of course we added a default `.gitignore` file to prevent your configuration setting and entry points to be committed in your *Git* history. That way you can have different configuration on development and on your production server without bothering about merge conflicts.

Generate JWT private and public keys

When using *composer create-project* command, you should have JWT secret and certificate automatically generated. If not, you can generate them using the following commands:

```
# Generate Symfony secrets
docker-compose exec -u www-data app bin/console secrets:generate-keys;
# Set a random passphrase for Application secret and JWT keys
docker-compose exec -u www-data app bin/console secrets:set APP_SECRET --random;
docker-compose exec -u www-data app bin/console secrets:set JWT_PASSPHRASE --
↪random;
# Use built-in command to generate your key pair
docker-compose exec -u www-data app bin/console lexik:jwt:generate-keypair;
```

Install database

```
# Create and migrate Roadiz database schema
docker-compose exec -u www-data app bin/console doctrine:migrations:migrate
# Migrate any existing data types
docker-compose exec -u www-data app bin/console app:migrate
# Install base Roadiz fixtures, default translation, roles and settings
docker-compose exec -u www-data app bin/console install
# Stop workers to force restart them
docker compose exec -u www-data app php bin/console messenger:stop-workers
# Clear cache
docker-compose exec -u www-data app bin/console cache:clear
# Create your admin account
docker-compose exec -u www-data app bin/console users:create -m username@roadiz.io␣
↪-b -s username
```

Then connect to `http://localhost:${YOUR_PORT}/rz-admin` to access your freshly-created Roadiz backoffice.

Note: If you setup Traefik on your local environment, you can reach your Roadiz app using your `domain.test` test domain name without specifying a non-default port. You have to change `HOSTNAME` dot-env variable and change your local DNS to point `domain.test` to `127.0.0.1`. The easiest way is to add `127.0.0.1 domain.test` to your `/etc/hosts` file.

Manual configuration

Roadiz is a full-stack Symfony application. It follows its configuration scheme as described in https://symfony.com/doc/5.4/configuration.html

Choose your inheritance model

Roadiz main feature is all about its polymorphic document model which is mapped on a relational database. This requires a challenging structure which can be lead to some performance bottlenecks when dealing with more than 20-30 node-types. So we made the data inheritance model configurable to allow switching to single_table scheme which will be more performant if you need lots of node-types. However *Single class* model will drop support for fields with the *same name but not the same type* because all node-type fields will be created in the **same SQL table**.

If you really need to mix field types, we advise you to keep the original joined table inheritance type which creates a dedicated SQL table for each node-type. *Joined table* inheritance can be very useful with a small number of node-type (max. 20) and very different fields. But its main drawback is that Roadiz needs to *LEFT JOIN* every node-type table for each node-source query, **unless you specify one node-type criteria**.

You can configure *Doctrine* strategy for NodesSources inheritance classes in `config/packages/roadiz_core.yaml`:

```
# config/packages/roadiz_core.yaml
roadiz_core:
    inheritance:
        # type: joined
        type: single_table
```

- Joined class inheritance: `joined`
- Single table inheritance: `single_table`

Warning: If you change this setting after creating content in your website, all node-sources data will be lost.

Themes (compatibility with v1.x)

Themes are statically registered into Roadiz configuration for better performances and delaying database usage. You have to register any front-end theme in your `config/packages/roadiz_compat.yaml` file. Theme priority is not handled here but in each of your themes by overriding static `$priority` value;

```
# config/packages/roadiz_compat.yaml
roadiz_compat:
    themes:
        -
            classname: \Themes\DefaultTheme\DefaultThemeApp
            hostname: '*'
            routePrefix: ''
        -
            classname: \Themes\FooBarTheme\FooBarThemeApp
            hostname: 'foobar.test'
            routePrefix: ''
```

You can define hostname specific themes and add a route-prefix for your routing. Defaults values are `'*'` for the *hostname* and `''` (empty string) for the route-prefix.

Warning: No theme configuration will lead into a 404 error on your website home page. But you will still have access to the back-office which is now hard-registered into Roadiz configuration.

Solr endpoint

Roadiz can use an *Apache Solr* search-engine to index nodes-sources. Add this to your *config/packages/roadiz_core.yaml* to link your CMS to your *Solr* server:

```
# config/packages/roadiz_core.yaml
roadiz_core:
    solr:
        endpoint:
            localhost:
                host: "localhost"
                port: "8983"
                path: "/"
                core: "mycore"
                timeout: 3
```

```
                username: ""
                password: ""
```

Roadiz CLI command can easily handle Solr index. Just type `./bin/console solr:check` to get more information.

Reverse proxy cache invalidation

Roadiz can request cache invalidation to external and internal cache proxies such as internal *Symfony* AppCache or a *Varnish* instance. If configured, Roadiz will create a `BAN` request to each configured proxy **when user clears back-office caches**, and it will create a `PURGE` request **on each node-source** *update event* using first reachable node-source URL.

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend:
            default:
                host: '%env(string:VARNISH_HOST)%'
                domainName: '%env(string:VARNISH_DOMAIN)%'
```

Note: Make sure you configured your external reverse proxy in order to receive and handle `BAN` and `PURGE` HTTP requests.

With API Platform you also need to configure `http_cache` invalidation section:

```
# config/packages/api_platform.yaml
api_platform:
    http_cache:
        invalidation:
            enabled: true
            varnish_urls: ['%env(VARNISH_URL)%']
```

Cloudflare proxy cache

If you are using Cloudflare as a reverse proxy cache, you can configure Roadiz to send requests to Cloudflare to purge all items or files (when editing a node-source). You need to gather following information:

- Cloudflare zone identifier
- Cloudflare API credentials (Bearer token or email + auth-key)

Then you can configure Roadiz with Bearer token:

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend: []
        cloudflare:
            zone: cloudflare-zone
            bearer: ~
```

Or with your Email and AuthKey:

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend: []
        cloudflare:
            zone: cloudflare-zone
```

```
            email: ~
            key: ~
```

Note: Roadiz uses *Purge all files* and *Purge Files by URL* entry points: https://api.cloudflare.com/#zone-purge-all-files which are available on all Cloudflare plans.

Entities paths

Roadiz uses *Doctrine* to map object entities to database tables.

```
# config/packages/doctrine.yaml
doctrine:
    orm:
        auto_generate_proxy_classes: true
        default_entity_manager: default
        entity_managers:
            # Put `logger` entity manager first to select it as default for Log␣
↪entity
            logger:
                naming_strategy: doctrine.orm.naming_strategy.underscore_number_
↪aware
                mappings:
                    ## Just sharding EM to avoid having Logs in default EM
                    ## and flushing bad entities when storing log entries.
                    RoadizCoreLogger:
                        is_bundle: false
                        type: attribute
                        dir: '%kernel.project_dir%/vendor/roadiz/core-bundle/src/
↪Logger/Entity'
                        prefix: 'RZ\Roadiz\CoreBundle\Logger\Entity'
                        alias: RoadizCoreLogger
            default:
                dql:
                    string_functions:
                        JSON_CONTAINS:␣
↪Scienta\DoctrineJsonFunctions\Query\AST\Functions\Mysql\JsonContains
                naming_strategy: doctrine.orm.naming_strategy.underscore_number_
↪aware
                auto_mapping: true
                mappings:
                    ## Keep RoadizCoreLogger to avoid creating different␣
↪migrations since we are using
                    ## the same database for both entity managers. Just sharding␣
↪EM to avoid
                    ## having Logs in default EM and flushing bad entities when␣
↪storing log entries.
                    RoadizCoreLogger:
                        is_bundle: false
                        type: attribute
                        dir: '%kernel.project_dir%/vendor/roadiz/core-bundle/src/
↪Logger/Entity'
                        prefix: 'RZ\Roadiz\CoreBundle\Logger\Entity'
                        alias: RoadizCoreLogger
                    RoadizCoreBundle:
                        is_bundle: true
                        type: attribute
                        dir: 'src/Entity'
                        prefix: 'RZ\Roadiz\CoreBundle\Entity'
                        alias: RoadizCoreBundle
                    RZ\Roadiz\Core:
```

```
                        is_bundle: false
                        type: attribute
                        dir: '%kernel.project_dir%/lib/Models/src/Core/
↪AbstractEntities'
                        prefix: 'RZ\Roadiz\Core\AbstractEntities'
                        alias: AbstractEntities
                    App\GeneratedEntity:
                        is_bundle: false
                        type: attribute
                        dir: '%kernel.project_dir%/src/GeneratedEntity'
                        prefix: 'App\GeneratedEntity'
                        alias: App\GeneratedEntity
                    App:
                        is_bundle: false
                        type: attribute
                        dir: '%kernel.project_dir%/src/Entity'
                        prefix: 'App\Entity'
                        alias: App
                    gedmo_loggable:
                        type: attribute
                        prefix: Gedmo\Loggable\Entity\MappedSuperclass
                        dir: "%kernel.project_dir%/vendor/gedmo/doctrine-
↪extensions/src/Loggable/Entity/MappedSuperclass"
                        alias: GedmoLoggableMappedSuperclass
                        is_bundle: false
```

Use `type: attribute` or `type: annotation` according to your Doctrine mapping type.

Configure mailer

Roadiz uses *Symfony Mailer* to send emails.

https://symfony.com/doc/5.4/mailer.html#transport-setup

Note: Pay attention that many external SMTP services (*Mandrill*, *Mailjet*...) only accept email from validated domains. So make sure that your application uses a known `From:` email sender not to be blacklisted or blocked by these services. If you need your emails to be replied to an anonymous address, use `ReplyTo:` header instead.

Images processing

Roadiz use Intervention Request Bundle to automatically create a lower quality version of your image if they are too big and offer on-the-fly image resizing and optimizing.

```
# config/packages/rz_intervention_request.yaml
parameters:
    env(IR_DEFAULT_QUALITY): '90'
    env(IR_MAX_PIXEL_SIZE): '1920'
    ir_default_quality: '%env(int:IR_DEFAULT_QUALITY)%'
    ir_max_pixel_size: '%env(int:IR_MAX_PIXEL_SIZE)%'

rz_intervention_request:
    driver: 'gd'
    default_quality: '%ir_default_quality%'
    max_pixel_size: '%ir_max_pixel_size%'
    cache_path: "%kernel.project_dir%/public/assets"
    files_path: "%kernel.project_dir%/public/files"
    jpegoptim_path: /usr/bin/jpegoptim
    pngquant_path: /usr/bin/pngquant
    subscribers: []
```

Additional *Intervention Request* subscribers

Any *Intervention Request* subscriber can be added to configuration with its `classname` and its constructor arguments. Here is an example with `WatermarkListener` which will print some text on all your images.

```
rz_intervention_request:
    # List additional Intervention Request subscribers
    subscribers:
        - class: "AM\\InterventionRequest\\Listener\\WatermarkListener"
          args:
                - 'Copyright 2017'
                - 3
                - 50
                - "#FF0000"
```

Use kraken.io to reduce drastically image sizes

Since you can add *Intervention Request* subscribers, we created a useful one that sends every images to kraken.io services to shrink them. Once you've configured it, do not forget to empty your caches **and** image caches to see changes.

```
rz_intervention_request:
    # List additional Intervention Request subscribers
    subscribers:
        - class: "AM\\InterventionRequest\\Listener\\KrakenListener"
          args:
                - "your-api-key"
                - "your-api-secret"
                - true
```

Warning: Take note that each generated image is sent to *kraken.io* servers. It can generate some overhead time on the first time you request an image.

Enable Two-factor authentication

Roadiz can use *Two-factor authentication* to secure your back-office access. You need to install `composer require roadiz/two-factor-bundle` and configure it in your `config/packages/scheb_2fa.yaml` and `config/packages/security.yaml` files.

See Two-factor authentication bundle documentation.

OpenID SSO authentication

Roadiz can use *OpenID* authentication to allow your users to log in with their Google account.

It supports 2 modes:

- **Requires local user: Users must have a local account to be able to log in with OpenID.** This is the default mode.
- **No local user required: Users can log in with OpenID without having a local account. A virtual** account will be created for them with their email address as username and roles listed in `granted_roles`. With this mode, you cannot use Preview mode as it requires a local user.

For both modes, you can restrict users to a specific domain with `hosted_domain` parameter.

```
# config/packages/roadiz_rozier.yaml
roadiz_rozier:
    open_id:
        # Verify User info in JWT at each login
        verify_user_info: false
```

```
        # Standard OpenID autodiscovery URL, required to enable OpenId login in␣
↪Roadiz CMS.
        discovery_url: '%env(string:OPEN_ID_DISCOVERY_URL)%'
        # For public identity providers (such as Google), restrict users emails by␣
↪their domain.
        hosted_domain: '%env(string:OPEN_ID_HOSTED_DOMAIN)%'
        # OpenID identity provider OAuth2 client ID
        oauth_client_id: '%env(string:OPEN_ID_CLIENT_ID)%'
        # OpenID identity provider OAuth2 client secret
        oauth_client_secret: '%env(string:OPEN_ID_CLIENT_SECRET)%'
        requires_local_user: false
        granted_roles:
            - ROLE_USER
            - ROLE_BACKEND_USER
            - ROLE_ACCESS_VERSIONS
            - ROLE_ACCESS_DOCTRINE_CACHE_DELETE
            - ROLE_ACCESS_DOCUMENTS
            - ROLE_ACCESS_DOCUMENTS_LIMITATIONS
            - ROLE_ACCESS_DOCUMENTS_DELETE
            - ROLE_ACCESS_DOCUMENTS_CREATION_DATE
            - ROLE_ACCESS_NODES
            - ROLE_ACCESS_NODES_DELETE
            - ROLE_ACCESS_NODES_SETTING
            - ROLE_ACCESS_NODES_STATUS
            - ROLE_ACCESS_REDIRECTIONS
            - ROLE_ACCESS_TAGS
            - ROLE_ACCESS_TAGS_DELETE
            - ROLE_ACCESS_CUSTOMFORMS
            - ROLE_ACCESS_CUSTOMFORMS_DELETE
            - ROLE_ACCESS_CUSTOMFORMS_RETENTION
            - ROLE_ACCESS_ATTRIBUTES
            - ROLE_ACCESS_ATTRIBUTES_DELETE
            - ROLE_ACCESS_NODE_ATTRIBUTES
            - ROLE_ACCESS_SETTINGS
            - ROLE_ACCESS_LOGS
            - ROLE_ACCESS_USERS
            - ROLE_ACCESS_USERS_DELETE
            - ROLE_ACCESS_GROUPS
            - ROLE_ACCESS_TRANSLATIONS
```

Console commands

Roadiz can be executed as a simple CLI tool using your SSH connection. This is useful to handle basic administration tasks with no need of backoffice administration.

```
./bin/console
```

If your system is not configured to have *php* located in `/usr/bin/php` use it this way:

```
php ./bin/console
```

Default command with no arguments will show you the available commands list. Each command has its own parameters. You can use the argument `--help` to get more information about each tool:

```
./bin/console install --help
```

We even made *Doctrine* CLI tools directly available from Roadiz Console. Be careful, these are powerful commands which can alter your database and make you lose precious data. Especially when you will need to update your database schema after a Theme or a Core update. **Always make a database back-up before any Doctrine operation**.

Upgrading

Note: **Always do a database backup before upgrading.** You can use the *mysqldump* or *pg_dump* tools to quickly export your database as a file.

- With a MySQL server: `mysqldump -u[user] -p[user_password] [database_name] > dumpfilename.sql`
- With a PostgreSQL server: `pg_dump -U [user] [database_name] -f dumpfilename.sql`

Use *Composer* to update dependencies or Roadiz itself with *Standard* or *Headless* editions, make sure that your Roadiz *version constraint* is set in your project `composer.json` file, then:

```
composer update -o;
```

Run database registered migrations (some migrations will be skipped according to your database type). Doctrine migrations are the default method to upgrade all none-node-type related entities:

```
bin/console doctrine:migrations:migrate;
```

In order to avoid losing sensible node-sources data. You should regenerate your node-source entities classes files:

```
bin/console generate:nsentities;
```

Then check if there is no pending SQL changes due to your Roadiz node-types, this should be addressed with a `doctrine:migrations:migrate` but you can check it with:

```
bin/console doctrine:schema:update --dump-sql;
# Upgrade node-sources tables if necessary
bin/console doctrine:schema:update --dump-sql --force;
```

Then, clear your app caches:

```
# Clear cache for each environment
bin/console cache:clear -e dev
bin/console cache:clear -e prod
bin/console cache:pool:clear cache.global_clearer
bin/console messenger:stop-workers
```

Note: If you are using a runtime cache like OPcache or APCu, you'll need to purge cache manually because it can't be done from a CLI interface as they are shared cache engines. As a last chance try, you can restart your `php-fpm` service.

Upgrading from Roadiz v2.1 to v2.2

Here is an extract for the Changelog

- Doctrine migrations are now the default method to upgrade all node-type related entities. You should run `bin/console doctrine:migrations:migrate` after updating your Roadiz dependencies.
- Roadiz updated to API Platform new version and Metadata scheme. You must rewrite your api resource YAML files to match new scheme. See API Platform documentation. You can remove any `ns_**.yml` api resource files then run `bin/console generate:api-resources` to generate them again. But any custom serialization groups will be lost.
- All node-type updates after Roadiz 2.2 will be versioned and **will generate a Doctrine migration file**. You may generate a Migration file with any existing node-type and add it without executing it if you want to keep a clean migration path, for new fresh website installs.

- `roadiz/models` entities path changed from `%kernel.project_dir%/vendor/roadiz/models/src/Roadiz/Core/AbstractEntities` to `%kernel.project_dir%/lib/Models/src/Core/AbstractEntities`
- `Logger` is now handled by a different entity-manager to avoid flushing non-valid entities when persisting log entries into database.

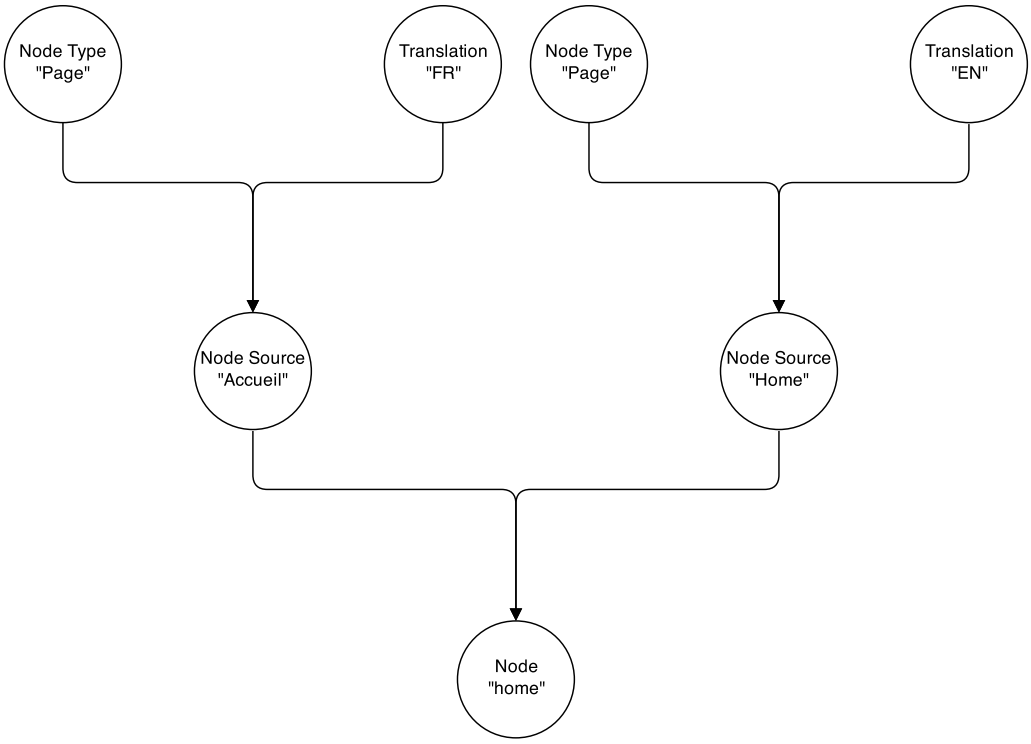
```
orm:
    auto_generate_proxy_classes: true
    default_entity_manager: default
    entity_managers:
        # Put `logger` entity manager first to select it as default for Log entity
        logger:
            naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
            mappings:
                ## Just sharding EM to avoid having Logs in default EM
                ## and flushing bad entities when storing log entries.
                RoadizCoreLogger:
                    is_bundle: false
                    type: attribute
                    dir: '%kernel.project_dir%/vendor/roadiz/core-bundle/src/
↪Logger/Entity'
                    prefix: 'RZ\Roadiz\CoreBundle\Logger\Entity'
                    alias: RoadizCoreLogger
        default:
            dql:
                string_functions:
                    JSON_CONTAINS:␣
↪Scienta\DoctrineJsonFunctions\Query\AST\Functions\Mysql\JsonContains
            naming_strategy: doctrine.orm.naming_strategy.underscore_number_aware
            auto_mapping: true
            mappings:
                ## Keep RoadizCoreLogger to avoid creating different migrations␣
↪since we are using
                ## the same database for both entity managers. Just sharding EM to␣
↪avoid
                ## having Logs in default EM and flushing bad entities when␣
↪storing log entries.
                RoadizCoreLogger:
                    is_bundle: false
                    type: attribute
                    dir: '%kernel.project_dir%/vendor/roadiz/core-bundle/src/
↪Logger/Entity'
                    prefix: 'RZ\Roadiz\CoreBundle\Logger\Entity'
                    alias: RoadizCoreLogger
                App:
                    is_bundle: false
                    type: attribute
                    dir: '%kernel.project_dir%/src/Entity'
                    prefix: 'App\Entity'
                    alias: App
                # ...
```

3.1.2 Node system

Node-types, nodes-sources and nodes

This part is the most important part of Roadiz. Quite everything in your website will be a node.

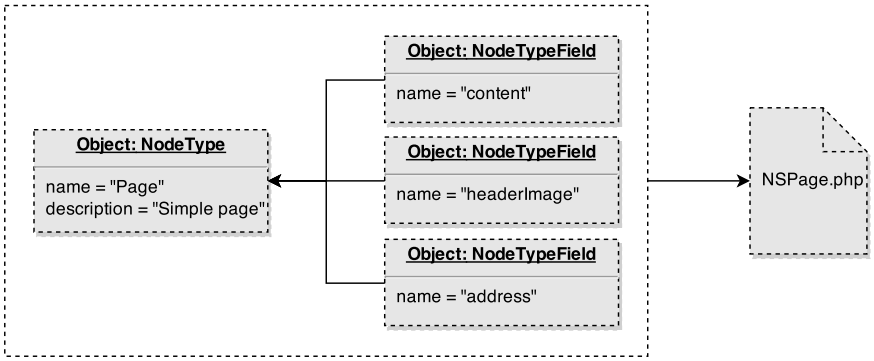
Let's check this simple node schema before explain it.



Now, it's time to explain how it's working!

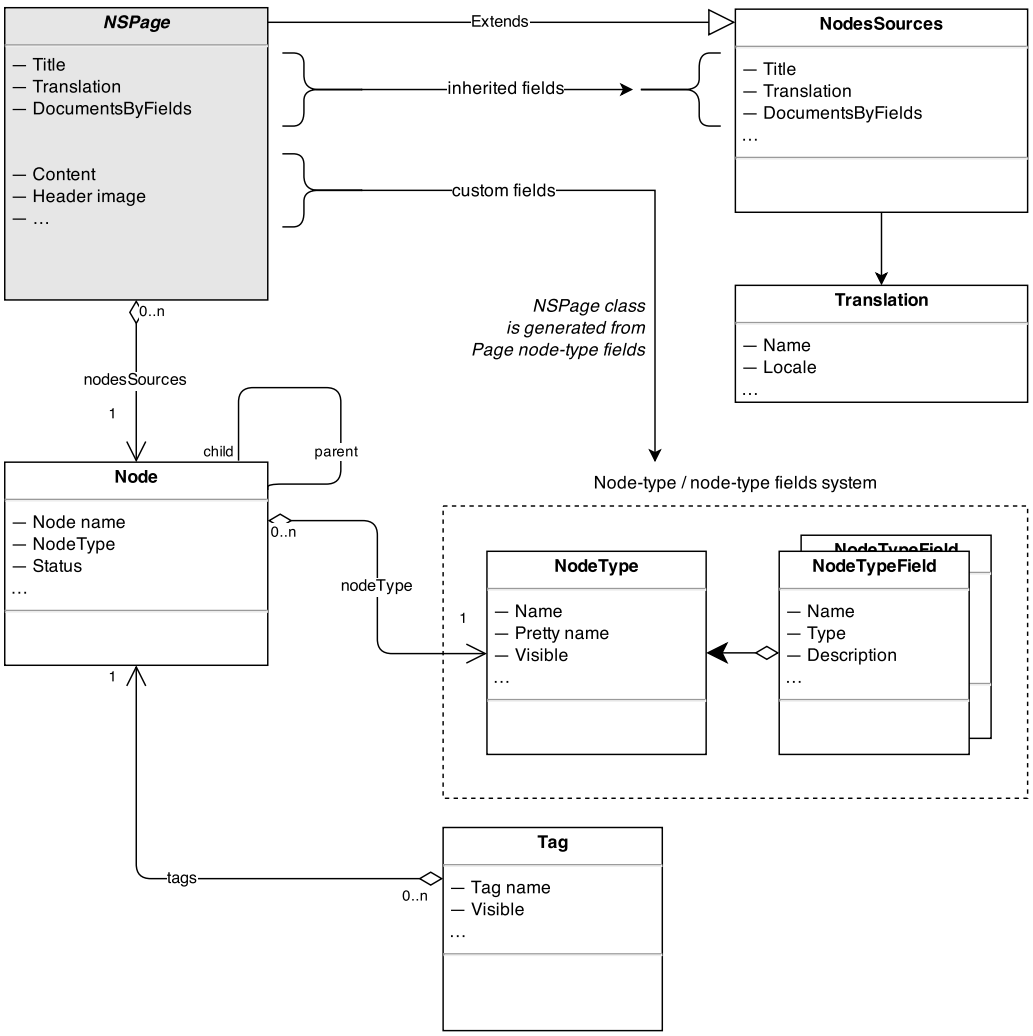
What is a Node-type

A node-type is the blueprint for your node-source. It will contain all fields that Roadiz will use to generate an extended node-source class.



For example, a node-type "Page" will contain "content" and "header image" fields. The "title" field is always available as it is hard-coded in `NodesSources` class. After saving your node-type, Roadiz generates a `NSPage` class which extends the `NodesSources` class. You will find it in the `gen-src/GeneratedNodeSources` (or `app/gen-src/GeneratedNodeSources` with *Roadiz Standard edition*). Then Roadiz calls *Doctrine* update tool to migrate your database schema. **Do not modify the generated class.** You'll have to update it by the backend interface.

Here is a schema to understand how node-types can define custom fields into node-sources:



Most of node-types management will be done in your backoffice interface. You will be able to create, update node-types objects and each of their node-type fields independently. But if you prefer, you can use CLI commands to create types and fields. With Roadiz CLI commands you get several tools to manage node-types. We really encourage you to check the commands with `--help` argument, as following:

```
bin/console nodetypes:add-fields
bin/console nodetypes:create
bin/console nodetypes:delete
bin/console nodetypes:list
```

Keep in mind that each node-type or node-type fields operation require a database update as Doctrine have to create a specific table per node-type. Do not forget to execute `bin/console doctrine:schema:update` tools to perform updates. It's very important to understand that *Doctrine* needs to see your node-types generated classes **before** upgrading database schema. If they don't exist, it won't able to create your custom types tables, or worst, it could delete existing data since *Doctrine* won't recognize specific tables.

Now let's have a look on node-sources.

Node-sources and translations

Once your node-type created, its definition is stored in database in `node_types` and `node_type_fields` tables. This informations will be only used to build your node-sources edition forms in backoffice and to build a custom database table.

Inheritance mapping

With Roadiz, each node-types data (called node-sources) is stored in a different table prefixed with `ns_`. When you create a *Page* node-type with 2 fields (*content* and *excerpt*), Roadiz tells Doctrine to build a `ns_page` table with 2 columns and one primary key column inherited from `nodes_sources` table. It's called *inheritance mapping*: your `ns_page` table extends `nodes_sources` table and when you are querying a *Page* from database, Doctrine mix the data coming from these 2 tables to create a complete node-source.

At the end your node-source *Page* won't contain only 2 fields but many more as `NodesSources` entity offers `title`, `metaTitle`, `metaDescription`, `metaKeywords` and others useful data-fields which can be used among all node-types.

Translations

Node-sources inheritance mapping is not only used to customize data but to make data translations available. As you saw in the first picture, each nodes can handle many node-sources, one per translation.

Node-type fields

Roadiz can handle many types of node-type fields. Here is a complete list:

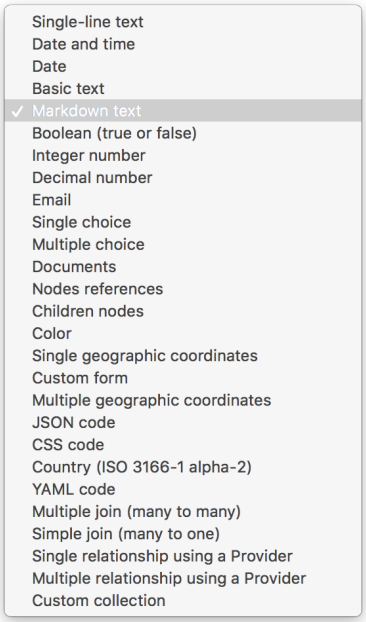
Note: *Title*, *meta-title*, *meta-description* and *keywords* are always available since they are stored directly inside `NodesSources` entity. Then you will be sure to always have a *title* no matter the node-type you are using.

Simple data

This following fields stores simple data in your custom node-source database table.

- Single-line text
- Date
- Date and time
- Basic text
- Markdown text
- Boolean
- Integer number
- Decimal number
- Email
- Color
- Single geographic coordinates
- Multiple geographic coordinates
- JSON code
- CSS code
- Country code (ISO 3166-1 alpha-2)
- YAML code
- Many to many join
- Many to one join
- Single relationship using a provider
- Multiple relationship using a provider

- Custom collection



Single and multiple geographic coordinates

Geographic coordinates are stored in JSON format in your database using GeoJSON schema:

- A single point will be stored as a GeoJSON *feature* in order to hold additional properties such as *zoom*,
- Multiple points will be stored as a GeoJSON *feature collection*

By default, Roadiz back-office uses *Leaflet* library with *Open Street Map* for tiles rendering and basic geo-coding features.

Markdown options

You can restrict Markdown fields buttons using the following YAML configuration:

```
allow_h2: false
allow_h3: false
allow_h4: false
allow_h5: false
allow_h6: false
allow_bold: false
allow_italic: false
allow_blockquote: false
allow_list: false
allow_nbsp: false
allow_nb_hyphen: false
allow_image: false
allow_return: false
allow_link: false
allow_hr: false
allow_preview: false
```

Virtual data

Virtual types do not really store data in node-source table. They display custom widgets in your editing page to link documents, nodes or custom-forms with your node-source.

- Documents

- Nodes references
- Custom form

Complex data

These fields types must be created with *default values* (comma separated) in order to display available default choices for "select-box" types:

- Single choice
- Multiple choices
- Children nodes

Children node field type is a special virtual field that will display a custom node-tree inside your editing page. You can add *quick-create* buttons by listing your node-types names in *default values* input, comma separated.

Universal fields

If you need a field to hold exactly the same data for all translations, you can set it as *universal*. For example for documents, numeric and boolean data that do not change from one language to another.

It will duplicate data at each save time from default translation to others. It will also hide the edit field from non-default translation to avoid confusion.

YAML field

When you use YAML field type, you get an additional method to return your code already parsed. If your field is named `data`, your methods will be generated in your *NSEntity* as `getData()` and `getDataAsObject()`.

- `getData()` method will return your YAML code as *string*.
- `getDataAsObject()` will return a mixed data,array or `stdObject` according to your code formatting. This method will throw a `\Symfony\Component\Yaml\Exception\ParseException` if your YAML code is not valid.

Many to many and Many to one joins

You can create custom relations between your node-source and whatever Doctrine entities in *Roadiz* or in your theme.

You must fill the *default values* field for these two types.

```
# Entity class name
classname: Themes\MyTheme\Entities\City
# Displayable is the method used to display entity name
displayable: getName
# Same as Displayable but for a secondary information
alt_displayable: getZipCode
# Searchable entity fields
searchable:
    - name
    - slug
orderBy:
    - field: slug
      direction: ASC
```

You can use a custom proxy entity to support persisting `position` on your relation. Roadiz will generate a one-to-many relationship with proxy entity instead of a many-to-many. In this scenario you are responsible for creating and migrating `Themes\MyTheme\Entities\PositionedCity` entity. If you are migrating from a non-proxied many-to-many relation, you should keep the same table and field names to keep data intact.

```
# Entity class name
classname: Themes\MyTheme\Entities\City
# Displayable is the method used to display entity name
displayable: getName
# Same as Displayable but for a secondary information
alt_displayable: getZipCode
# Searchable entity fields
searchable:
    - name
    - slug
# This order will only be used for explorer
orderBy:
    - field: slug
      direction: ASC
# Use a proxy entity
proxy:
    classname: Themes\MyTheme\Entities\PositionedCity
    self: nodeSource
    relation: city
    # This order will preserve position
    orderBy:
        - field: position
          direction: ASC
```

Single and multiple provider

The generic provider type allow you to fetch every data you want through a `Provider` class in your theme. This can be really useful if you need to fetch items from an external API and to reference them in your nodes-sources.

Imagine that you want to link your page with an *Instagram* post. You'll have to create a class that extends `Themes\Rozier\Explorer\AbstractExplorerProvider` and configure it in your field:

```
classname: Themes\MyTheme\Provider\ExternalApiProvider
```

This provider will implement `getItems`, `getItemsById` and other methods from `ExplorerProviderInterface` in order to be able to display your *Instagram* posts in Roadiz explorer widget and to find your selected items back. Each *Instagram* post will be wrapped in a `Themes\Rozier\Explorer\AbstractExplorerItem` that will map your custom data to the right fields to be showed in Roadiz back-office.

You'll find an implementation example in Roadiz with `Themes\Rozier\Explorer\SettingsProvider` and `Themes\Rozier\Explorer\SettingExplorerItem`. These classes do not fetch data from an API but from your database using `EntityListManager`.

Single and multiple provider types can accept additional options too. If you want to make your provider configurable at runtime you can pass `options` in your field configuration.

```
classname: Themes\MyTheme\Provider\ExternalApiProvider
options:
    - name: user
      value: me
    - name: access_token
      value: xxxxx
```

Then you must override your provider' `configureOptions` method to add which options are allowed.

```
use Symfony\Component\OptionsResolver\OptionsResolver;

/**
 * @param OptionsResolver $resolver
 */
public function configureOptions(OptionsResolver $resolver)
```

```
{
    $resolver->setDefaults([
        'page' => 1,
        'search' => null,
        'itemPerPage' => 30,
        // add more default options here
        'user' => 'me',
    ]);
    // You can required options
    $resolver->setRequired('access_token');
}
```

Custom collection

Last but not least, you can create a custom collection field to store read-only data using a dedicated *Symfony* `AbstractType`.

You must fill the *default values* field for this type:

```
# AbstractType class name
entry_type: Themes\MyTheme\Form\FooBarType
```

You must understand that *custom collection* data will be stored as JSON array in your database. So you won't be able to query your node-source using this data.

In your `FooBarType`, you'll be able to use *Symfony* standard fields types and **Roadiz** non-virtual fields too such as `MarkdownType`, `JsonType`, `YamlType`.

Handling nodes and their hierarchy

By default, if you use Entities API methods or traversing Twig filters, Roadiz will automatically handle security parameters such as `node.status` and `preview` mode.

```
// Secure method to get node-sources
// Implicitly check node.status
$this->nodeSourceApi->getBy([
    'node.nodeType' => $blogPostType,
    'translation' => $translation,
], [
    'publishedAt' => 'DESC'
]);
```

This first code snippet is using *Node-source API*. This will automatically check if current user is logged-in and if preview mode is *ON* to display or not *unpublished nodes*.

```
// Insecure method to get node-sources
// Doctrine raw method will get all node-sources
$this->managerRegistry->getRepository(NSBlogPost::class)->findBy([], [
    'publishedAt' => 'DESC',
    'translation' => $translation,
]);
```

This second code snippet uses standard Doctrine *Entity Manager* to directly grab node-sources by their entity class. This method does not check any security and will return every node-sources, **even unpublished, archived and deleted ones**.

Hierarchy

To traverse node-sources hierarchy, the easier method is to use *Twig* filters on your `nodeSource` entity. Filters will implicitly set `translation` from origin node-source.

```
{% set children = nodeSource|children %}
{% set nextSource = nodeSource|next %}
{% set prevSource = nodeSource|previous %}
{% set parent = nodeSource|parent %}

{% set children = nodeSource|children({
    'node.visible': true
}) %}
```

Warning: All these filters will take care of publication status and translation, **but not publication date-time neither visibility**.

```
{% set children = nodeSource|children({
    'node.visible': true,
    'publishedAt': ['>=', date()],
}, {
    'publishedAt': 'DESC'
}) %}

{% set nextVisible = nodeSource|next({
    'node.visible': true
}) %}
```

If you need to traverse node-source graph from your controllers you can use the *Entity API*. Moreover, Nodes-sources API allows you to filter using custom criteria if you choose a specific `NodeType`.

```
$children = $this->nodeSourceApi->getBy([
    'node.parent' => $nodeSource,
    'node.visible' => true,
    'publishedAt' => ['>=', new \DateTime()],
    'translation' => $nodeSource->getTranslation(),
],[
    'publishedAt' => 'DESC'
]);
```

Warning: Browsing your node graph (calling children or parents) could be very greedy and unoptimized if you have lots of node-types. Internally *Doctrine* will *inner-join* every nodes-sources tables to perform polymorphic hydration. So, make sure you filter your queries by one `NodeType` as much as possible with `nodeSourceApi` and `node.nodeType` criteria.

```
// Here Doctrine will only join NSPage table to NodesSources
$children =$this->nodeSourceApi->getBy([
    'node.nodeType' => $this->nodeTypesBag->get('Page'),
    'node.parent' => $nodeSource,
    'node.visible' => true,
    'publishedAt' => ['>=', new \DateTime()],
    'translation' => $nodeSource->getTranslation(),
],[
    'publishedAt' => 'DESC'
]);
```

Visibility

There are two parameters that you must take care of in your themes and your controllers, because they are not mandatory in all website cases:

- Visibility

- Publication date and time

For example, *publication date and time* won't be necessary in plain text pages and not timestampable contents. But we decided to add it directly in `NodesSources` entity to be able to filter and order with this field in Roadiz back-office. This was not possible if you manually create your own `publishedAt` as a node-type field.

> **Warning:** Pay attention that *publication date and time* (`publishedAt`) and visibility (`node.visible`) **does not prevent** your node-source from being viewed if you did not explicitly forbid access to its controller. This field is not deeply set into Roadiz security mechanics.
>
> If you need so, make sure that your node-type controller checks these two fields and throws a `ResourceNotFoundException` if they're not satisfied.

```
class BlogPostController extends MyAwesomeTheme
{
    public function indexAction(
        Request $request,
        Node $node = null,
        TranslationInterface $translation = null
    ) {
        $this->prepareThemeAssignation($node, $translation);

        $now = new DateTime("now");
        if (!$nodeSource->getNode()->isVisible() ||
            $nodeSource->getPublishedAt() < $now) {
            throw new ResourceNotFoundException();
        }

        return $this->render(
            'types/blogpost.html.twig',
            $this->assignation
        );
    }
}
```

Publication workflow

Each Node state is handled by a *Workflow* to switch between the following 5 states:

States

- `Node::DRAFT`
- `Node::PENDING`
- `Node::PUBLISHED`
- `Node::ARCHIVED`
- `Node::DELETED`

Transitions

- review
- reject
- publish
- archive
- unarchive

- delete
- undelete

You cannot changes a Node status directly using its *setter*, you must use Roadiz main *registry* to perform transition. This can prevent unwanted behaviours and you can track changes with events and guards:

```
$nodeWorkflow = $this->workflowRegistry->get($node);
if ($nodeWorkflow->can($node, 'publish')) {
    $nodeWorkflow->apply($node, 'publish');
}
```

Generating paths and url

You can use `generateUrl()` in your controllers to get a node-source' path or url. In your Twig template, you can use `path` method as described in Twig section: twig-generate-paths.

```
use Symfony\Cmf\Component\Routing\RouteObjectInterface;

class BlogPostController extends MyAwesomeTheme
{
    public function indexAction(
        Request $request,
        Node $node = null,
        TranslationInterface $translation = null
    ) {
        $this->prepareThemeAssignation($node, $translation);

        // Generate a path for current node-source
        $path = $this->generateUrl(
            RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
            [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource]
        );

        // Generate an absolute URL for current node-source
        $absoluteUrl =  $this->generateUrl(
            RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
            [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource],
            UrlGeneratorInterface::ABSOLUTE_URL
        );
    }
}
```

Overriding default node-source path generation

You can override default node-source path generation in order to use `{{ path() }}` method in your *Twig* templates but with a custom logic. For example, you have a `Link` node-type which purpose only is to link to an other node in your website. When you call *path* or *URL* generation on it, you should prefer getting its linked node path, so you can listen to `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent:class` event and stop propagation to return your linked node path instead of your *link* node path.

```
use GeneratedNodeSources\NSLink;
use Symfony\Component\EventDispatcher\EventDispatcherInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent;

class LinkPathGeneratingEventListener implements EventSubscriberInterface
{
    public static function getSubscribedEvents(): array
    {
        return [
```

```
            NodesSourcesPathGeneratingEvent::class => ['onLinkPathGeneration']
        ];
    }

    /**
     * @param NodesSourcesPathGeneratingEvent $event
     * @param string                          $eventName
     * @param EventDispatcherInterface        $dispatcher
     */
    public function onLinkPathGeneration(
        NodesSourcesPathGeneratingEvent $event,
        $eventName,
        EventDispatcherInterface $dispatcher
    ) {
        $nodeSource = $event->getNodeSource();

        if ($nodeSource instanceof NSLink) {
            if (filter_var($nodeSource->getExternalUrl(), FILTER_VALIDATE_URL)) {
                /*
                 * If editor linked to an external link
                 */
                $event->stopPropagation();
                $event->setComplete(true);
                $event->setContainsScheme(true); // Tells router not to prepend␣
↪protocol and host to current URL
                $event->setPath($nodeSource->getExternalUrl());
            } elseif (count($nodeSource->getNodeReferenceSources()) > 0 &&
                null !== $linkedSource = $nodeSource->
↪getNodeReferenceSources()[0]) {
                /*
                 * If editor linked to an internal page through a node reference
                 */
                /** @var FilterNodeSourcePathEvent $subEvent */
                $subEvent = clone $event;
                $subEvent->setNodeSource($linkedSource);
                /*
                 * Dispatch a path generation again for linked node-source.
                 */
                $dispatcher->dispatch($subEvent);
                /*
                 * Fill main event with sub-event data
                 */
                $event->setPath($subEvent->getPath());
                $event->setComplete($subEvent->isComplete());
                $event->setParameters($subEvent->getParameters());
                $event->setContainsScheme($subEvent->containsScheme());
                // Stop propagation AFTER sub-event was dispatched not to prevent␣
↪it to perform.
                $event->stopPropagation();
            }
        }
    }
}
```

3.1.3 Building headless websites using API

Since Roadiz v2, headless development is default and is the most powerful way to build reactive websites and applications. Roadiz is built on API Platform, and it exposes all main entities as API Resources using DTO (data transfer objects)

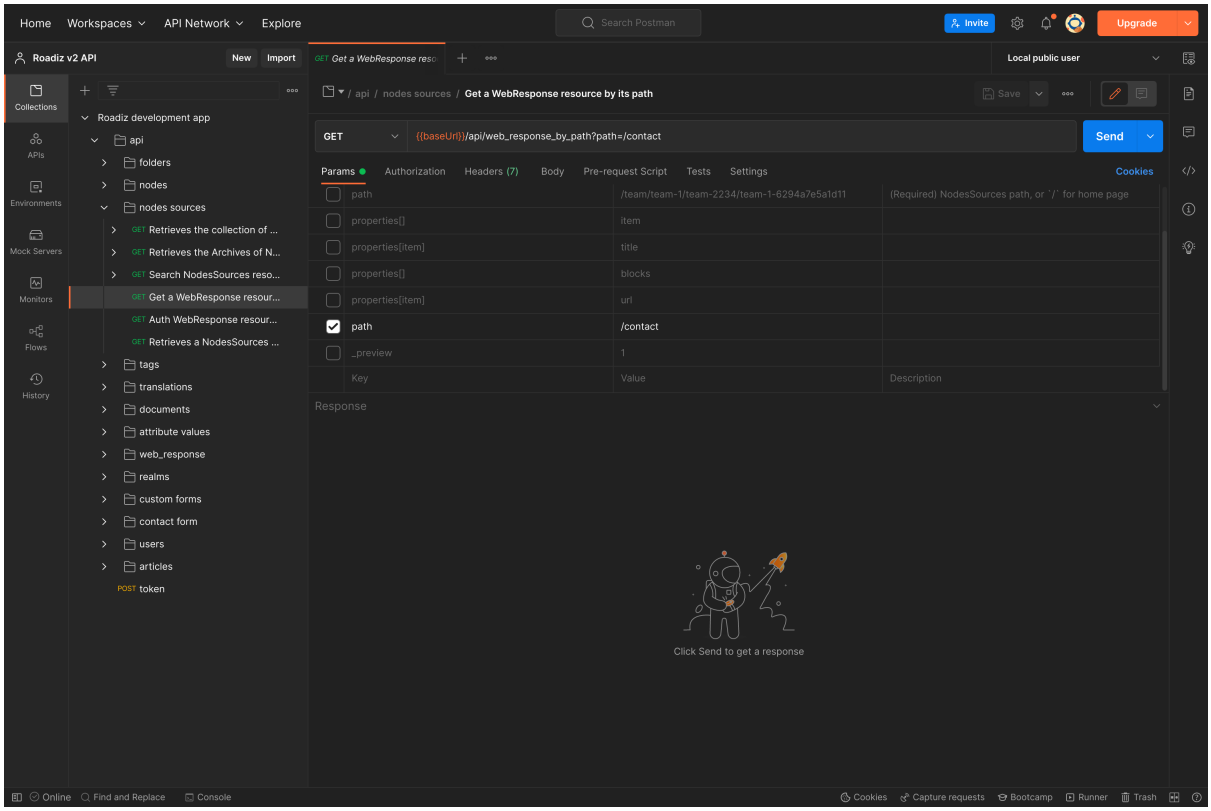


Fig. 3.1: Consuming Roadiz API with Postman application is a great way to explore and test REST calls for your frontend app

WebResponse concept

A REST-ful API will expose collection and item entry-points for each resource. But in both case, you need to know your resource type or your resource identifier **before** executing your API call. Roadiz introduces a special resource named **WebResponse** which can be called using a `path` query param in order to reduce as much as possible API calls and address N+1 problem.

```
GET /api/web_response_by_path?path=/contact
```

API will expose a WebResponse single item containing:

- An item
- Item breadcrumbs
- Head object
- Item blocks tree-walker
- Item realms
- and if blocks are hidden by Realm configuration

Note: Roadiz *WebResponse* is used in Rezo Zero Nuxt Starter to populate all data during the `asyncData()` routine in `_.vue` page

```
{
    "@context": "/api/contexts/WebResponse",
    "@id": "/api/web_response_by_path?path=/contact",
    "@type": "WebResponse",
    "item": {
```

```
        "@id": "/api/pages/7",
        "@type": "Page",
        "content": "Magni deleniti ut eveniet. Aliquam aut et excepturi vitae␣
↪placeat molestiae. Molestiae asperiores nihil sed temporibus quibusdam. Non␣
↪magnam fuga at. sdf",
        "subTitle": null,
        "overTitle": null,
        "headerImage": [],
        "test": null,
        "pictures": [],
        "nodeReferences": [],
        "stickytest": false,
        "sticky": false,
        "customForm": [],
        "title": "Contact",
        "publishedAt": "2021-09-10T15:56:00+02:00",
        "metaTitle": "",
        "metaKeywords": "",
        "metaDescription": "",
        "users": [],
        "node": {
            "@type": "Node",
            "@id": "/api/nodes/7",
            "visible": true,
            "position": 3,
            "tags": []
        },
        "slug": "contact",
        "url": "/contact"
    },
    "breadcrumbs": {
        "@type": "Breadcrumbs",
        "@id": "_:14750",
        "items": []
    },
    "head": {
        "@type": "NodesSourcesHead",
        "@id": "_:14679",
        "googleAnalytics": null,
        "googleTagManager": null,
        "matomoUrl": null,
        "matomoSiteId": null,
        "siteName": "Roadiz dev website",
        "metaTitle": "Contact - Roadiz dev website",
        "metaDescription": "Contact, Roadiz dev website",
        "policyUrl": null,
        "mainColor": null,
        "facebookUrl": null,
        "instagramUrl": null,
        "twitterUrl": null,
        "youtubeUrl": null,
        "linkedinUrl": null,
        "homePageUrl": "/",
        "shareImage": null
    },
    "blocks": [],
    "realms": [],
    "hidingBlocks": false
}
```

Override WebResponse block walker

Imagine you have a block (*ArticleFeedBlock*) which should list latest news (*Article*). You can use tree-walker mechanism to fetch latest news and expose them as if they were children of your article feed block. This requires to create a custom definition:

```php
<?php

declare(strict_types=1);

namespace App\TreeWalker\Definition;

use App\GeneratedEntity\NSArticle;
use App\GeneratedEntity\NSArticleFeedBlock;
use Doctrine\ORM\Tools\Pagination\Paginator;
use RZ\Roadiz\CoreBundle\Api\TreeWalker\NodeSourceWalkerContext;
use RZ\Roadiz\CoreBundle\Entity\NodesSources;
use RZ\TreeWalker\Definition\ContextualDefinitionTrait;
use RZ\TreeWalker\Definition\StoppableDefinition;
use RZ\TreeWalker\WalkerInterface;

final class ArticleFeedBlockDefinition implements StoppableDefinition
{
    use ContextualDefinitionTrait;

    public function isStoppingCollectionOnceInvoked(): bool
    {
        return true;
    }

    /**
     * @param NodesSources $source
     * @param WalkerInterface $walker
     * @return array
     * @throws \Exception
     */
    public function __invoke(NodesSources $source, WalkerInterface $walker): array
    {
        if ($this->context instanceof NodeSourceWalkerContext) {
            $this->context->getStopwatch()->start(self::class);
            if (!$source instanceof NSArticleFeedBlock) {
                throw new \InvalidArgumentException('Source must be instance of ' .
↪ NSArticleFeedBlock::class);
            }

            $criteria = [
                'node.visible' => true,
                'publishedAt' => ['<=', new \DateTime()],
                'translation' => $source->getTranslation(),
                'node.nodeType' => $this->context->getNodeTypesBag()->get('Article
↪')
            ];

            // Prevent Article feed to list root Article again
            $root = $walker->getRoot()->getItem();
            if ($root instanceof NSArticle) {
                $criteria['id'] = ['!=', $root->getId()];
            }

            if (null !== $source->getNode() && \count($source->getNode()->
↪getTags()) > 0) {
                $criteria['tags'] = $source->getNode()->getTags();
                $criteria['tagExclusive'] = true;
```

```
            }

            $count = (int) ($source->getListingCount() ?? 4);

            $children = $this->context->getNodeSourceApi()->getBy($criteria, [
                'publishedAt' => 'DESC'
            ], $count);


            if ($children instanceof Paginator) {
                $iterator = $children->getIterator();
                if ($iterator instanceof \ArrayIterator) {
                    $children = $iterator->getArrayCopy();
                } else {
                    throw new \RuntimeException('Unexpected iterator type');
                }
            }

            $this->context->getStopwatch()->stop(self::class);

            return $children;
        }
        throw new \InvalidArgumentException('Context should be instance of ' .␣
↪NodeSourceWalkerContext::class);
    }
}
```

Then create a definition factory which will be injected using Symfony autoconfigure tag `roadiz_core.tree_walker_definition_factory`.

`roadiz_core.tree_walker_definition_factory` tag must include a `classname` attribute which will be used to match your definition factory with the right node source class.

```
<?php

declare(strict_types=1);

namespace App\TreeWalker\Definition;

use App\GeneratedEntity\NSArticleFeedBlock;
use RZ\Roadiz\CoreBundle\Api\TreeWalker\Definition\DefinitionFactoryInterface;
use RZ\TreeWalker\WalkerContextInterface;
use Symfony\Component\DependencyInjection\Attribute\AutoconfigureTag;

#[AutoconfigureTag(
    name:'roadiz_core.tree_walker_definition_factory',
    attributes: ['classname' => NSArticleFeedBlock::class]
)]
final class ArticleFeedBlockDefinitionFactory implements DefinitionFactoryInterface
{
    public function create(WalkerContextInterface $context, bool $onlyVisible =␣
↪true): callable
    {
        return new ArticleFeedBlockDefinition($context);
    }
}
```

This way, all tree-walkers will be able to use your custom definition anytime a `NSArticleFeedBlock` is encountered.

You can debug all registered definition factories using `bin/console debug:container --tag=roadiz_core.tree_walker_definition_factory` command.

Retrieve common content

Now that we can fetch each page data, we need to get all unique content for building Menus, Homepage reference, headers, footers, etc. We could extend our _WebResponse_ to inject theses common data to each request, but it would bloat HTTP responses, and affect API performances.

For these common content, you can create a `/api/common_content` API endpoint in your project which will fetched only once in your frontend application.

```
# config/api_resources/common_content.yml
App\Api\Model\CommonContent:
    operations:
        getCommonContent:
            class: ApiPlatform\Metadata\Get
            method: 'GET'
            uriTemplate: '/common_content'
            read: false
            controller: App\Controller\GetCommonContentController
            pagination_enabled: false
            normalizationContext:
                enable_max_depth: true
                pagination_enabled: false
                groups:
                    - get
                    - common_content
                    - web_response
                    - walker
                    - walker_level
                    - children
                    - children_count
                    - nodes_sources_base
                    - nodes_sources_default
                    - urls
                    - blocks_urls
                    - tag_base
                    - translation_base
                    - document_display
                    - document_folders
```

Note: Keep in mind that `/api/common_content` endpoint uses `nodes_sources_base` normalization group which **will only include essential node sources data**. You can add more groups to include more data, such as `nodes_sources_default` or `nodes_sources_cta` if you grouped some fields into a *CTA* label.

Then create you own custom resource to hold your menus tree-walkers and common content. Tree-walkers will be created using `RZ\Roadiz\CoreBundle\Api\TreeWalker\TreeWalkerGenerator` service. TreeWalkerGenerator will create a `App\TreeWalker\MenuNodeSourceWalker` instance for each node source of type `Menu` located on your website root.

```
<?php

declare(strict_types=1);

namespace App\Controller;

use App\Api\Model\CommonContent;
use App\TreeWalker\MenuNodeSourceWalker;
use Doctrine\Persistence\ManagerRegistry;
use RZ\Roadiz\Core\AbstractEntities\TranslationInterface;
use RZ\Roadiz\CoreBundle\Api\Model\NodesSourcesHeadFactoryInterface;
use RZ\Roadiz\CoreBundle\Api\TreeWalker\TreeWalkerGenerator;
use RZ\Roadiz\CoreBundle\Preview\PreviewResolverInterface;
```

```php
use RZ\Roadiz\CoreBundle\Repository\TranslationRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\RequestStack;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
use Symfony\Component\Routing\Exception\ResourceNotFoundException;

final class GetCommonContentController extends AbstractController
{
    private RequestStack $requestStack;
    private ManagerRegistry $managerRegistry;
    private NodesSourcesHeadFactoryInterface $nodesSourcesHeadFactory;
    private PreviewResolverInterface $previewResolver;
    private TreeWalkerGenerator $treeWalkerGenerator;

    public function __construct(
        RequestStack $requestStack,
        ManagerRegistry $managerRegistry,
        NodesSourcesHeadFactoryInterface $nodesSourcesHeadFactory,
        PreviewResolverInterface $previewResolver,
        TreeWalkerGenerator $treeWalkerGenerator
    ) {
        $this->requestStack = $requestStack;
        $this->managerRegistry = $managerRegistry;
        $this->nodesSourcesHeadFactory = $nodesSourcesHeadFactory;
        $this->previewResolver = $previewResolver;
        $this->treeWalkerGenerator = $treeWalkerGenerator;
    }

    public function __invoke(): ?CommonContent
    {
        try {
            $request = $this->requestStack->getMainRequest();
            $translation = $this->getTranslationFromRequest($request);

            $resource = new CommonContent();

            $request?->attributes->set('data', $resource);
            $resource->head = $this->nodesSourcesHeadFactory->createForTranslation(
↪$translation);
            $resource->home = $resource->head->getHomePage();
            $resource->menus = $this->treeWalkerGenerator->
↪getTreeWalkersForTypeAtRoot(
                'Menu',
                MenuNodeSourceWalker::class,
                $translation,
                3
            );
            return $resource;
        } catch (ResourceNotFoundException $exception) {
            throw new NotFoundHttpException($exception->getMessage(), $exception);
        }
    }

    protected function getTranslationFromRequest(?Request $request):␣
↪TranslationInterface
    {
        $locale = null;

        if (null !== $request) {
            $locale = $request->query->get('_locale');

            /*
```

```
             * If no _locale query param is defined check Accept-Language header
             */
            if (null === $locale) {
                $locale = $request->getPreferredLanguage($this->
↪getTranslationRepository()->getAllLocales());
            }
        }
        /*
         * Then fallback to default CMS locale
         */
        if (null === $locale) {
            $translation = $this->getTranslationRepository()->findDefault();
        } elseif ($this->previewResolver->isPreview()) {
            $translation = $this->getTranslationRepository()
                ->findOneByLocaleOrOverrideLocale((string) $locale);
        } else {
            $translation = $this->getTranslationRepository()
                ->findOneAvailableByLocaleOrOverrideLocale((string) $locale);
        }
        if (null === $translation) {
            throw new NotFoundHttpException('No translation for locale ' .
↪$locale);
        }
        return $translation;
    }

    protected function getTranslationRepository(): TranslationRepository
    {
        $repository = $this->managerRegistry->
↪getRepository(TranslationInterface::class);
        if (!$repository instanceof TranslationRepository) {
            throw new \RuntimeException(
                'Translation repository must be instance of ' .
                TranslationRepository::class
            );
        }
        return $repository;
    }
}
```

Then, the following resource will be exposed:

```
{
    "@context": "/api/contexts/CommonContent",
    "@id": "/api/common_content",
    "@type": "CommonContent",
    "home": {
        "@id": "/api/pages/11",
        "@type": "Page",
        "content": null,
        "image": [],
        "title": "Accueil",
        "publishedAt": "2022-04-12T16:24:00+02:00",
        "node": {
            "@type": "Node",
            "@id": "/api/nodes/10",
            "visible": true,
            "tags": []
        },
        "slug": "accueil",
        "url": "/fr"
    },
    "menus": {
```

```
        "mainMenuWalker": {
            "@type": "MenuNodeSourceWalker",
            "@id": "_:3341",
            "children": [],
            "childrenCount": 0,
            "item": {
                "@id": "/api/menus/2",
                "@type": "Menu",
                "title": "Menu principal",
                "publishedAt": "2022-04-12T00:39:00+02:00",
                "node": {
                    "@type": "Node",
                    "@id": "/api/nodes/1",
                    "visible": false,
                    "tags": []
                },
                "slug": "main-menu"
            },
            "level": 0,
            "maxLevel": 3
        },
        "footerMenuWalker": {
            "@type": "MenuNodeSourceWalker",
            "@id": "_:2381",
            "children": [],
            "childrenCount": 0,
            "item": {
                "@id": "/api/menus/3",
                "@type": "Menu",
                "linkInternalReference": [],
                "title": "Menu du pied de page",
                "publishedAt": "2022-04-12T11:18:12+02:00",
                "node": {
                    "@type": "Node",
                    "@id": "/api/nodes/2",
                    "visible": false,
                    "tags": []
                },
                "slug": "footer-menu"
            },
            "level": 0,
            "maxLevel": 3
        },
        "footerWalker": {
            "@type": "AutoChildrenNodeSourceWalker",
            "@id": "_:2377",
            "children": [],
            "childrenCount": 0,
            "item": {
                "@id": "/api/footers/16",
                "@type": "Footer",
                "content": "",
                "title": "Pied de page",
                "publishedAt": "2022-04-12T19:02:47+02:00",
                "node": {
                    "@type": "Node",
                    "@id": "/api/nodes/15",
                    "visible": false,
                    "tags": []
                },
                "slug": "footer"
            },
            "level": 0,
```

```
            "maxLevel": 3
        }
    },
    "head": {
        "@type": "NodesSourcesHead",
        "@id": "_:14679",
        "googleAnalytics": null,
        "googleTagManager": null,
        "matomoUrl": null,
        "matomoSiteId": null,
        "siteName": "Roadiz dev website",
        "metaTitle": "Contact - Roadiz dev website",
        "metaDescription": "Contact, Roadiz dev website",
        "policyUrl": null,
        "mainColor": null,
        "facebookUrl": null,
        "instagramUrl": null,
        "twitterUrl": null,
        "youtubeUrl": null,
        "linkedinUrl": null,
        "homePageUrl": "/",
        "shareImage": null
    }
}
```


Exposing node-types

All resources configuration files are located in `./config/api_resources` folder.

Here is an example of API resource for a `Page` node-type, you'll find default `operations``plus a special ``getByPath` operation which allow overriding **WebResponse** serialization groups.

```
App\GeneratedEntity\NSPage:
    types:
        - Page
    operations:
        ApiPlatform\Metadata\GetCollection:
            method: GET
            shortName: Page
            normalizationContext:
                enable_max_depth: true
                groups:
                    - nodes_sources_base
                    - nodes_sources_default
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
                    - document_thumbnails
                    - document_display_sources
                    - nodes_sources_images
                    - nodes_sources_boolean
        _api_page_archives:
            method: GET
            class: ApiPlatform\Metadata\GetCollection
            shortName: Page
            uriTemplate: /pages/archives
            extraProperties:
                archive_enabled: true
            openapiContext:
                summary: 'Retrieve all Page ressources archives months and years'
        ApiPlatform\Metadata\Get:
```

```
            method: GET
            shortName: Page
            normalizationContext:
                groups:
                    - nodes_sources
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
                    - document_thumbnails
                    - document_display_sources
                    - nodes_sources_images
                    - nodes_sources_boolean
        getByPath:
            method: GET
            class: ApiPlatform\Metadata\Get
            uriTemplate: /web_response_by_path
            read: false
            controller:␣
↪RZ\Roadiz\CoreBundle\Api\Controller\GetWebResponseByPathController
            normalizationContext:
                pagination_enabled: false
                enable_max_depth: true
                groups:
                    - nodes_sources
                    - node_listing
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
                    - document_thumbnails
                    - document_display_sources
                    - nodes_sources_images
                    - nodes_sources_boolean
                    - web_response
                    - walker
                    - walker_level
                    - walker_metadata
                    - meta
                    - children
            openapiContext:
                tags:
                    - WebResponse
                summary: 'Get a resource by its path wrapped in a WebResponse␣
↪object'
                description: 'Get a resource by its path wrapped in a WebResponse'
                parameters:
                    - { type: string, name: path, in: query, required: true,␣
↪description: 'Resource path, or `/` for home page', schema: { type: string } }
```

To automatically generate your resources YAML configuration files, execute the following CLI command:

```
bin/console generate:api-resources
```

Serialization groups

Roadiz CMS uses `symfony/serializer` to perform JSON serialization over any objects, especially *Doctrine* entities.

id Serialize every entity `id`.

timestamps Serialize every date-timed entity `createdAt` and `updatedAt` fields.

position Serialize every entity `position` fields.

color Serialize every entity `color` fields.

nodes_sources Serialize entities in a `NodesSources` context (all fields).

nodes_sources_base Serialize entities in a `NodesSources` context, but with essential information.

nodes_sources_documents Serialize documents linked to a `NodesSources` for each virtual field.

nodes_sources_default Serialize `NodesSources` fields not contained in any **group**.

nodes_sources_‘‘group‘‘ Custom serialization groups are created according to your node-typ fields groups. For example, if you set a field to a `link` group, `nodes_sources_link` serialization group will be automatically generated for this field. *Be careful*, Roadiz will use groups *canonical names* to generate serialization groups, it can mix _ and -.

node Serialize entities in a `Node` context.

node_children Serialize Nodes with their `children`.

node_attributes Serialize Nodes with their `attribute values`.

node_listing Serialize Nodes and NodesSources with their `listing children options`. Adds `listingSortOptions` field to node-sources, with for example `{ "node.position": "ASC" }` to be able to fetch children nodes in a the same order as in the back-office. If the current serialized node-source' parent is hiding its children, `listingSortOptions` will be contain parent's `listingSortOptions` instead.

tag Serialize Tag.

tag_base Serialize Tags with minimum information.

tag_parent Serialize Tags with their `parent`.

tag_children Serialize Tags with their `children`, do not use with `tag_parent` group.

node_type Serialize entities in a `NodeType` context.

attribute Serialize entities in a `Attribute` context.

attribute_documents Serialize documents linked to a `Attribute` for each virtual field.

custom_form Serialize entities in a `CustomForm` context.

document Serialize entities in a `Document` context.

document_display Serialize `Document` information required for displaying them.

document_private Serialize `Document` privacy information

document_display_sources Serialize `Document` information required for displaying alternative sources (audio, video).

document_thumbnails Serialize `Document` first available thumbnail.

document_folders Serialize `Document` information required for displaying attached **visible** folders.

document_folders_all Serialize **all** `Document` attached folders, even not visible ones.

folder Serialize entities in a `Folder` context.

translation Serialize entities in a `Translation` context.

translation_base Serialize `Translation` information required for displaying them.

setting Serialize entities in a `Setting` context.

setting_group Serialize entities in a `SettingGroup` context.

user Serialize entities in a `User` context.

user_group Serialize User entity with its groups.

user_role Serialize User entity with its roles.

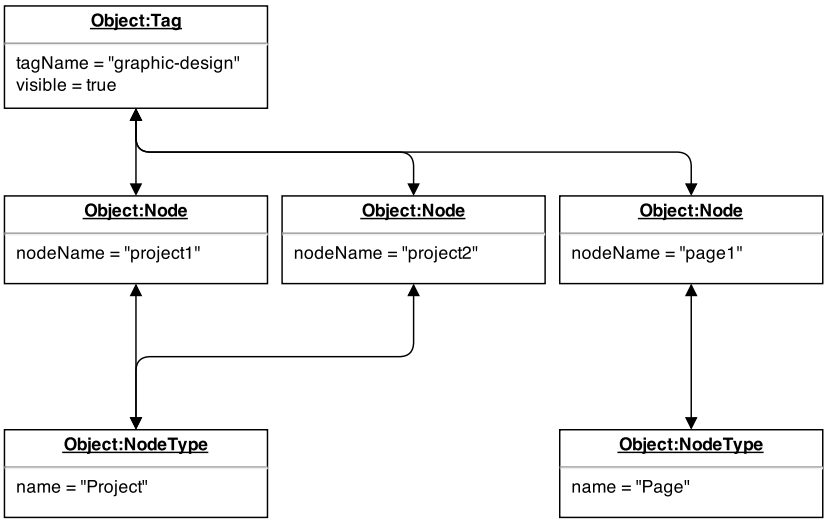
user_personal Serialize User entity with its personal information.

user_identifier Serialize User entity with its identifier (may be a personal information).

We recommend using Rezo Zero Nuxt starter to build your frontend applications. This starter is built to use Roadiz API and relies on dynamic routing and supports API redirections.

3.1.4 Tag system

Nodes are essentially hierarchical entities. So we created an entity to link nodes between them no matter where/what they are. Tags are meant as *tag* nodes, we couldn't be more explicit. But if you didn't understand here is a schema:

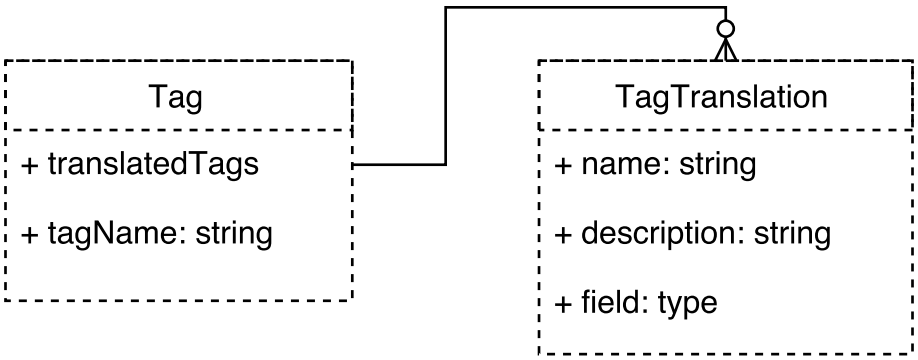


You can see that tags can gather heterogeneous nodes coming from different types (pages and projects). Tags can be used to display a category-navigation on your theme or to simply tidy your backoffice node database.

Did you notice that `Tags` are related to `Nodes` entities, not `NodesSources`? We thought that it would be easier to manage that way not to forget to tag a specific node translation. It means that you won't be able to differentiate tag two `NodesSources`, if you absolutely need to, we encourage you to create two different nodes.

Translate tags

You will notice that tags work the same way as nodes do. By default, *tags names* can't contain special characters in order to be used in URLs. So we created `TagTranslation` entities which stand for Tag's sources:



In that way you will be able to translate your tags for each available languages and link documents to them.

Tag hierarchy

In the same way as *Nodes* work, tags can be nested to create *tag groups*.

Exposing tags in API

When using API Platform data transfer objects, Tags are ready-to-use with translations set-up on `name` and `description` fields:

```
{
    "@type": "Tag",
    "@id": "/api/tags/6",
    "slug": "event",
    "name": "Event",
    "description": null,
    "color": "#000000",
    "visible": true,
    "documents": [],
    "parent": {
        "@type": "Tag",
        "@id": "/api/tags/3",
        "slug": "type",
        "name": "Type",
        "description": null,
        "color": "#000000",
        "visible": true,
        "documents": [],
        "parent": null
    }
}
```

Displaying node-source tags with Twig

Tag translations are already set up to track your current locale if you fetched them using `|tags` *Twig* filter. Simply use `getTranslatedTags()->first()` Tag method to use them in your templates.

```
{% set tags = nodeSource|tags %}

<ul>
{% for tag in tags %}
    {% set tagTranslation = tag.translatedTags.first %}
    <li id="{{ tag.tagName }}">{{ tagTranslation.name }}</li>
{% endfor %}
</ul>
```

Tags translations documents

Documents can be linked to your tag translations. They will be different for each translation, so make sure to synchronize them manually if you want to use the same document for all translations.

They are available with `getDocuments()` method and will be ordered by **position only**.

Imagine, you want to link a PDF document for each of your tags, you can create a download link as described below:

```
{% set tags = nodeSource|tags %}
<ul>
{% for tag in tags %}
    {% set tagTranslation = tag.translatedTags.first %}
    <li id="{{ tag.tagName }}">
        <p>{{ tagTranslation.name }}</p>
        {% if tagTranslation.documents[0] %}
            <a href="{{ tagTranslation.documents[0]|url }}" class="tag-document">{
↪% trans %}download_tag_pdf{% endtrans %}</a>
        {% endif %}
    </li>
```

```
{% endfor %}
</ul>
```

3.1.5 Documents system

Storing documents elsewhere...

Storing documents outside of your web-server is a good practice for many reasons:

- it allows to scale your application easily
- it allows to use a CDN to deliver your documents
- it allows to use a dedicated storage service (like Amazon S3) to store your documents

Documents system is based on *Flysystem*, a filesystem abstraction layer. It allows to store documents on local filesystem, Amazon S3, Google Cloud Storage, Rackspace Cloud Storage, Dropbox, FTP server, etc.

If you want to override default configuration, you can create a `config/packages/flysystem.yaml` file in your project and you must declare 3 storages:

- `documents_public.storage`: used to store public documents (accessible by anyone)
- `documents_private.storage`: used to store private documents (accessible only by authenticated users, or custom access control)
- `intervention_request.storage`: only used by Image processing system to store assets files

Following example shows how to configure Flysystem to use *Scaleway* Object Storage (S3 compatible) service:

```
# config/packages/flysystem.yaml
# Read the documentation at https://github.com/thephpleague/flysystem-bundle/blob/
↪master/docs/1-getting-started.md
services:
    scaleway_public_client:
        class: 'AsyncAws\SimpleS3\SimpleS3Client'
        arguments:
            -  endpoint: '%env(SCALEWAY_STORAGE_ENDPOINT)%'
               accessKeyId: '%env(SCALEWAY_STORAGE_ID)%'
               accessKeySecret: '%env(SCALEWAY_STORAGE_SECRET)%'
               region: '%env(SCALEWAY_STORAGE_REGION)%'
    # Private client must be different for allowing copy across file systems.
    scaleway_private_client:
        class: 'AsyncAws\SimpleS3\SimpleS3Client'
        arguments:
            -  endpoint: '%env(SCALEWAY_STORAGE_ENDPOINT)%'
               accessKeyId: '%env(SCALEWAY_STORAGE_ID)%'
               accessKeySecret: '%env(SCALEWAY_STORAGE_SECRET)%'
               region: '%env(SCALEWAY_STORAGE_REGION)%'

flysystem:
    storages:
        documents_public.storage:
            adapter: 'asyncaws'
            visibility: 'public'
            options:
                client: 'scaleway_public_client'
                bucket: '%env(SCALEWAY_STORAGE_BUCKET)%'
                prefix: 'testing-public-files'
        documents_private.storage:
            adapter: 'asyncaws'
            visibility: 'private'
            options:
                client: 'scaleway_private_client'
                bucket: '%env(SCALEWAY_STORAGE_BUCKET)%'
```

```
                prefix: 'testing-private-files'
        intervention_request.storage:
            adapter: 'asyncaws'
            visibility: 'public'
            options:
                client: 'scaleway_public_client'
                bucket: '%env(SCALEWAY_STORAGE_BUCKET)%'
                prefix: 'testing-public-files'
```

Exposing documents in API

When using API Platform data transfer objects, Documents are ready-to-use with translations set-up on `name` and `description` fields. Made sure to configure your API operations with at least `document_display` serialization group:

```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxx/my_image.jpg",
    "type": "image",
    "mimeType": "image/jpeg",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": "#141414",
    "imageWidth": 1000,
    "imageHeight": 750,
    "mediaDuration": 0,
    "copyright": "© John Doe",
    "externalUrl": null,
    "processable": true,
    "thumbnail": null,
    "alt": "This is an image"
}
```

Expose document thumbnails

Thumbnails are exposed by default for each document, it is useful when documents are not displayable: PDF, native video, ZIP, etc:

```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxx/img_2004_framed_1080p_2000.webm",
    "type": "video",
    "mimeType": "video/webm",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": null,
    "imageWidth": 1920,
    "imageHeight": 1080,
    "mediaDuration": 14,
    "copyright": null,
    "externalUrl": null,
    "processable": false,
    "thumbnail": {
        "@type": "Document",
        "@id": "/api/documents/xxxxx",
```

```
        "relativePath": "xxxxxxx/img_2004_framed_1080p_2000.png",
        "type": "image",
        "mimeType": "image/png",
        "name": null,
        "description": null,
        "embedId": null,
        "embedPlatform": null,
        "imageAverageColor": "#917357",
        "imageWidth": 2662,
        "imageHeight": 1504,
        "mediaDuration": 0,
        "copyright": null,
        "externalUrl": null,
        "processable": true,
        "thumbnail": null,
        "alt": "img_2004_framed_1080p_2000.png"
    },
    "alt": "img_2004_framed_1080p_2000.webm"
}
```


Expose document alternative sources

Alternative sources are not serialized by default for performance matters, but you can enable them in your project. Add `document_display_sources` serialization group to your resource configuration.


```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.webm",
    "type": "video",
    "mimeType": "video/webm",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": null,
    "imageWidth": 1920,
    "imageHeight": 1080,
    "mediaDuration": 14,
    "copyright": null,
    "externalUrl": null,
    "processable": false,
    "thumbnail": {
        "@type": "Document",
        "@id": "/api/documents/xxxxx",
        "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.png",
        "type": "image",
        "mimeType": "image/png",
        "name": null,
        "description": null,
        "embedId": null,
        "embedPlatform": null,
        "imageAverageColor": "#917357",
        "imageWidth": 2662,
        "imageHeight": 1504,
        "mediaDuration": 0,
        "copyright": null,
        "externalUrl": null,
        "processable": true,
        "thumbnail": null,
        "alt": "img_2004_framed_1080p_2000.png"
    },
```

```
    "altSources": [
        {
            "@type": "Document",
            "@id": "/api/documents/xxxxx",
            "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.mp4",
            "type": "video",
            "mimeType": "video/mp4",
            "name": null,
            "description": null,
            "embedId": null,
            "embedPlatform": null,
            "imageAverageColor": null,
            "imageWidth": 0,
            "imageHeight": 0,
            "mediaDuration": 0,
            "copyright": null,
            "externalUrl": null,
            "processable": false,
            "thumbnail": null,
            "alt": "img_2004_framed_1080p_2000.mp4"
        }
    ],
    "alt": "img_2004_framed_1080p_2000.webm"
}
```


Expose document folders

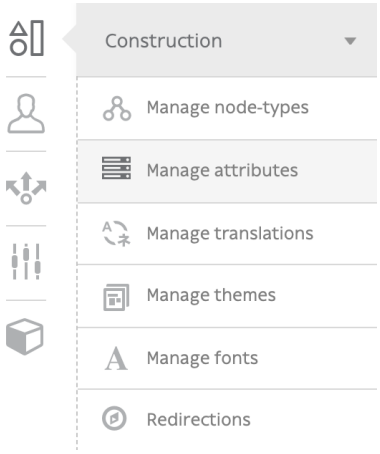
Document folders are not serialized by default for performance matters, but you can enable them in your project. Add `document_folders` serialization group to your resource configuration.


```
{
    "@type": "Document",
    "@id": "/api/documents/3436",
    "relativePath": "xxxxxxxx/youtube_wplj0yxcnwk.jpg",
    "type": "image",
    "mimeType": "image/jpeg",
    "name": "Shirine - Bande annonce",
    "description": "",
    "embedId": "wPlj0YxCNwk",
    "embedPlatform": "youtube",
    "imageAverageColor": "#2d2426",
    "imageWidth": 200,
    "imageHeight": 113,
    "mediaDuration": 0,
    "copyright": "Opéra de Lyon (https://www.youtube.com/user/OperadeLyon)",
    "externalUrl": null,
    "processable": true,
    "thumbnail": null,
    "folders": [
        {
            "@type": "Folder",
            "@id": "/api/folders/20",
            "slug": "danse",
            "name": "Danse",
            "visible": true
        },
        {
            "@type": "Folder",
            "@id": "/api/folders/31",
            "slug": "opera-inside",
            "name": "Opera-inside",
            "visible": false
```

```
        }
    ],
    "alt": "Shirine - Bande annonce"
}
```

3.1.6 Attributes

Attributes are entities meant to qualify features on other entities, such as *nodes*. The main difference between *tags* and *attributes* is that you can set a value for each attributed *node*.



The attribute “Color” can be set to “red” for one node and set to “green” for an other one.

Okay, but now what is the difference between *attributes* and node-type fields? Not so much because node-type fields describe your node’ features too, but they are fixed and defined by the developer. Once your node-type fields are created, you have to implement your feature in your *Twig* templates, and translate it in your XLF files too.

Attributes are meant to be created and added by editors so they can use them in your website lifecycle without needing any further development.

From the developer perspective, attributes are just a collection of entities to be displayed in a loop. Then your editor can create new ones and be sure they will be displayed without any additional development.

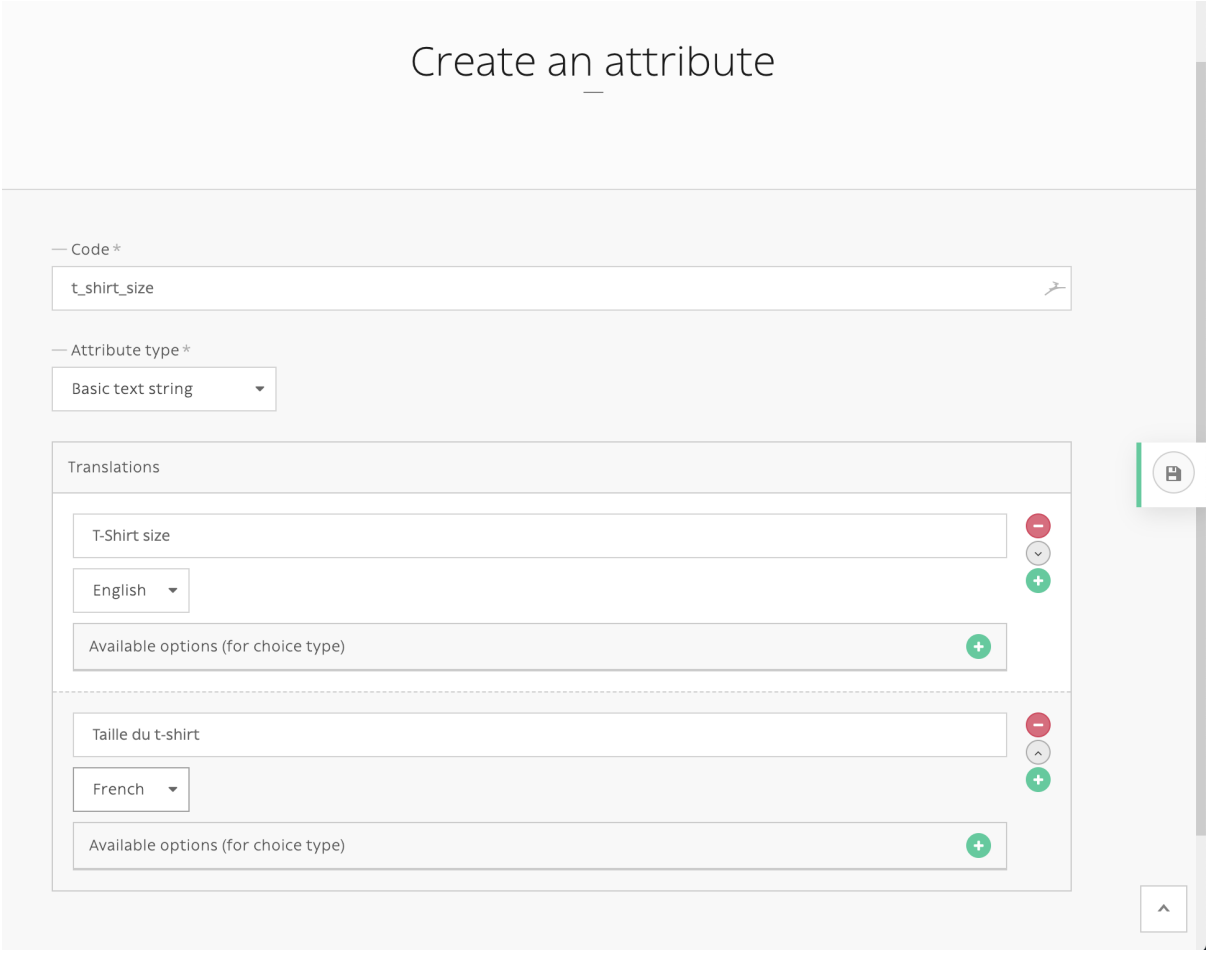
```
<ul class="block-attributes">
    {% for attributeValueTranslation in nodeSource|attributes %}
        <li>
            {% if attributeValueTranslation.attribute.documents|length %}
                {% for document in attributeValueTranslation.attribute.documents %}
                    <figure>{{ document|display }}</figure>
                {% endfor %}
            {% endif %}
            <strong>{{ attributeValueTranslation|attribute_label(translation) }}:</
↪strong>
            {% if attributeValueTranslation is datetime %}
                {{ attributeValueTranslation.value|format_datetime('medium', 'short
↪', locale=app.request.locale) }}
            {% elseif attributeValueTranslation is date %}
                {{ attributeValueTranslation.value|format_date('medium',␣
↪locale=app.request.locale) }}
            {% elseif attributeValueTranslation is country %}
                {{ attributeValueTranslation.value|country_name(request.locale) }}
            {% else %}
                {{ attributeValueTranslation.value }}
            {% endif%}
        </li>
    {% endfor %}
</ul>
```

If you grouped your attributes, you can use `grouped_attributes` filter instead:

```
<ul class="block-attributes">
    {% for item in nodeSource|grouped_attributes %}
        <li>
            {% if item.group %}
                <strong>{{ item.group|attribute_group_label(translation) }}</
↪strong>
            {% endif %}
            <ul>
                {% for attributeValueTranslation in item.attributeValues %}
                    <li>
                        {% if attributeValueTranslation.attribute.documents|length
↪%}
                            {% for document in attributeValueTranslation.attribute.
↪documents %}
                                <figure>{{ document|display }}</figure>
                            {% endfor %}
                        {% endif %}
                        <strong>{{ attributeValueTranslation|attribute_
↪label(translation) }}:</strong>
                        {% if attributeValueTranslation is datetime %}
                            {{ attributeValueTranslation.value|format_datetime(
↪'medium', 'short', locale=app.request.locale) }}
                        {% elseif attributeValueTranslation is date %}
                            {{ attributeValueTranslation.value|format_date('medium
↪', locale=app.request.locale) }}
                        {% elseif attributeValueTranslation is country %}
                            {{ attributeValueTranslation.value|country_
↪name(request.locale) }}
                        {% else %}
                            {{ attributeValueTranslation.value }}
                        {% endif%}
                    </li>
                {% endfor %}
            </ul>

        </li>
    {% endfor %}
</ul>
```

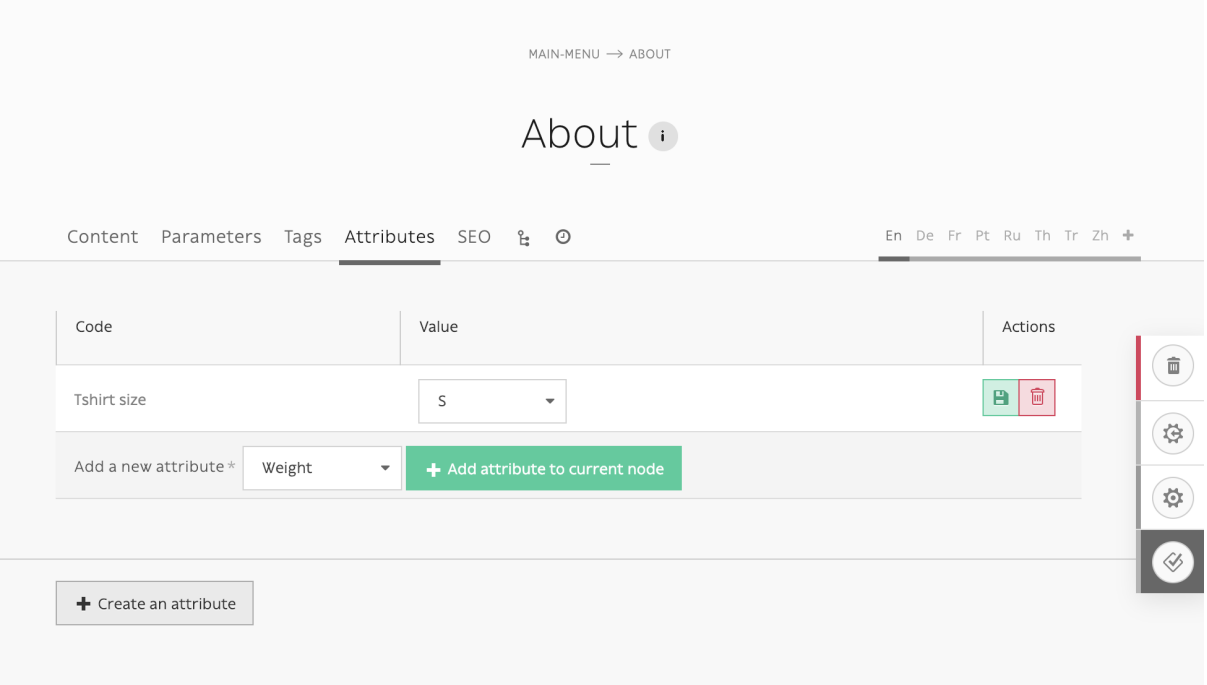
Attributes types



- *String*
- *Date*
- *Date and time*
- *Single choice*: choice among defined options in your attribute
- *Boolean*
- *Integer*
- *Decimal*
- *Email*
- *Color*
- *Country*: ISO 2-letters country code

Add attributes to nodes

Attribute section is available for any node in any translations.



Twig extension

Several filters and tests are available to ease up templating with *attributes*:

Filters

- `attributes`: same as `node_source_attributes()` method, get all available attributes from a `NodesSources`.
- `grouped_attributes`: same as `node_source_grouped_attributes()` method, get all available attributes from a `NodesSources` and gather them into their **group**.
- `attribute_label(translation)`: get attribute translated `label` or `code` if not translated.
- `attribute_group_label(translation)`: get attribute group translated `name` or `canonicalName` if not translated.

Tests

- `datetime`
- `date`
- `country`
- `boolean`
- `choice`
- `enum`

3.1.7 Forms

Roadiz uses *Symfony* forms logic and API. However, we made ready-made contact and custom forms builders to ease up your development and even make form-building available for your website editors.

Building contact forms

With Roadiz you can easily create simple contact forms with `ContactFormManager` class. Your controller has a convenient shortcut to create this manager with `$this->createContactFormManager()` method.

If you want to add your own fields, you can use the manager' form-builder with `$contactFormManager->getFormBuilder();`. Then add your field using standard *Symfony* form syntax. Do not forget to use *Constraints* to handle errors.

One contact-form for one action

Here is an example to create your contact form in your controller action.

```
use Symfony\Component\Validator\Constraints\File;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

// ...
// Create contact-form manager and add 3 default fields.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields();
/*
 * (Optional) Add custom fields...
 */
$formBuilder = $contactFormManager->getFormBuilder();
$formBuilder->add('callMeBack', CheckboxType::class, [
        'label' => 'call.me.back',
        'required' => false,
    ])
    ->add('document', FileType::class, [
        'label' => 'document',
        'required' => false,
        'constraints' => [
            new File([
                'maxSize' => $contactFormManager->getMaxFileSize(),
                'mimeTypes' => $contactFormManager->getAllowedMimeTypes(),
            ]),
        ]
    ])
    ->add('send', SubmitType::class, [
        'label' => 'send.contact.form',
    ]);

/*
 * This is the most important point. handle method will perform form
 * validation and send email.
 *
 * Handle method should return a Response object if everything is OK.
 */
if (null !== $response = $contactFormManager->handle()) {
    return $response;
}

$form = $contactFormManager->getForm();
```

In this example, we used `withDefaultFields` method which add automatically `email`, `name` and `message` fields with right validation constraints. This method is optional and you can add any field you want manually, just keep in mind that you should always ask for an `email`.

Then in your contact page Twig template:

```
{#
 # Display contact errors
 #}
{% for label, messages in app.flashes(['warning', 'error']) %}
   {% for message in messages %}
       <p class="alert alert-{{ label }}">
```

```
            {{- message -}}
        </p>
    {% endfor %}
{% endfor %}
{#
 # Display contact form
 #}
{% form_theme contactForm '@MyTheme/forms.html.twig' %}
{{ form(contactForm) }}
```

Using contact-form in *block* controllers

If you want to use *contact-forms* in blocks instead of a full page, you will need to make your redirection response **bubble** through *Twig* render. The only way to stop Twig is to **throw an exception** and to pass your Redirect or Json response within your Exception.

Roadiz makes this possible with `RZ\Roadiz\CoreBundle\Exception\ForceResponseException`. For example, in a `App/Controllers/ContactBlockController`, instead of returning the `contactFormManager` response, you will have to throw a `ForceResponseException` with it as an argument.

```
// ./src/Controllers/ContactBlockController.php

use RZ\Roadiz\CoreBundle\Exception\ForceResponseException;

...
// Create contact-form manager and add 3 default fields.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields();

if (null !== $response = $contactFormManager->handle()) {
    /*
     * Force response to bubble through Twig rendering process.
     */
    throw new ForceResponseException($response);
}

$form = $contactFormManager->getForm();
```

Then, in your *master* controller (i.e. `PageController`), `render` method will automatically catch your *ForceResponseException* exception in order to extract the forced response object. Then it will return your response instead of your page twig rendered output.

Alter email configuration after form submit

If you want to customize emails sent with form data, you can use Symfony form events to change the contact form manager options: The following example alters the email subject to add the user email and makes the subject unique for the receiver.

```
$formBuilder->addEventListener(FormEvents::SUBMIT, function (FormEvent $event) use␣
↪($contactFormManager) {
   $contactFormManager->setSubject($contactFormManager->getSubject() . ': ' .
↪$event->getForm()->get('email')->getData());
});
```

You can also use this behaviour to change dynamically the contact form receiver after the user chose it in a select box input.

Note: You can read more about form events at https://symfony.com/doc/current/form/events.html

Securing your form with *Google reCAPTCHA*

Roadiz can seamlessly use *Google reCAPTCHA* to secure your contact form against robots. All you need to do is to register on https://www.google.com/recaptcha/ to ask for a *sitekey* and a *secret*. Once you've got these two keys, add them to your `.env.local` or Symfony secrets.

- `APP_RECAPTCHA_PRIVATE_KEY`
- `APP_RECAPTCHA_PUBLIC_KEY`

Then, just use `withGoogleRecaptcha()` method on your contact-form manager.

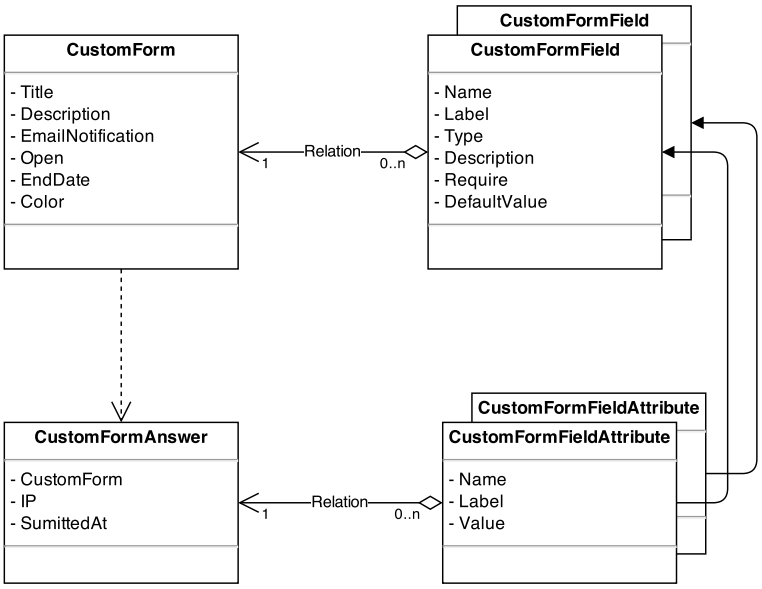
```
// Create contact-form manager, add 3 default fields and add a reCAPTCHA.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields()
                           ->withGoogleRecaptcha();
```

Do not forget to add recaptcha form-template and to embed google's javascript.

```
 {# In your theme' forms.html.twig file #}
 {% block recaptcha_widget -%}
    <input id="my-form-recaptcha" type="hidden" name="{{ form.vars.name }}" />
    <script src="https://www.google.com/recaptcha/api.js?render={{ configs.
↪publicKey }}"></script>
    <script>
        /*
         * Google Recaptcha v3
         * @see https://developers.google.com/recaptcha/docs/v3
         */
        (function() {
            if (!window.grecaptcha) {
                console.warn('Recaptcha is not loaded');
            }
            var form = document.getElementById('my-form');
            form.addEventListener('submit', function (event) {
                event.preventDefault();
                window.grecaptcha.ready(function() {
                    window.grecaptcha.execute('{{ configs.publicKey }}', {action:
↪'submit'}).then(function(token) {
                        var input = document.getElementById('my-form-recaptcha');
                        if (input) {
                            input.value = token;
                        }
                        form.submit()
                    });
                });
            });
        })();
    </script>
{%- endblock recaptcha_widget %}
```

Building custom forms

Building a custom form looks like building a node but it is a lot simpler! Let's have a look at structure image.



After creating a custom form, you add some question. The questions are the CustomFormField type.

The answer is saved in two entities:

- in CustomFormAnswer
- in CustomFormFieldAttribute

The CustomFormAnswer will store the IP and the submitted time. While question answer will be in CustomFormFieldAttribute with the CustomFormAnswer id and the CustomFormField id.

Exposing a custom form in your API

Custom-form can be filled in a headless context, using _definition_ and _post_ endpoints:

```
GET {{baseUrl}}/api/custom_forms/:id/definition
```

Custom form definition is a **JSON form schema** meant to give your frontend application a recipe to build a HTML form:

```
{
    "title": "",
    "type": "object",
    "properties": {
        "subject": {
            "type": "string",
            "title": "Subject",
            "attr": {
                "data-group": null,
                "placeholder": null
            },
            "description": "Est aut quas eum error architecto.",
            "propertyOrder": 1
        },
        "email": {
            "type": "string",
            "title": "Email",
            "attr": {
                "data-group": null,
```

```
                "placeholder": null
            },
            "description": "Email address",
            "widget": "email",
            "propertyOrder": 2
        },
        "test": {
            "title": "TEST",
            "type": "object",
            "properties": {
                "message": {
                    "type": "string",
                    "title": "Message",
                    "attr": {
                        "data-group": "TEST",
                        "placeholder": null
                    },
                    "widget": "textarea",
                    "propertyOrder": 1
                },
                "fichier": {
                    "type": "string",
                    "title": "File",
                    "attr": {
                        "data-group": "TEST",
                        "placeholder": null
                    },
                    "widget": "file",
                    "propertyOrder": 2
                }
            },
            "required": [
                "fichier"
            ],
            "attr": {
                "data-group-wrapper": "test"
            },
            "propertyOrder": 3
        }
    },
    "required": [
        "subject",
        "email",
        "test"
    ]
}
```

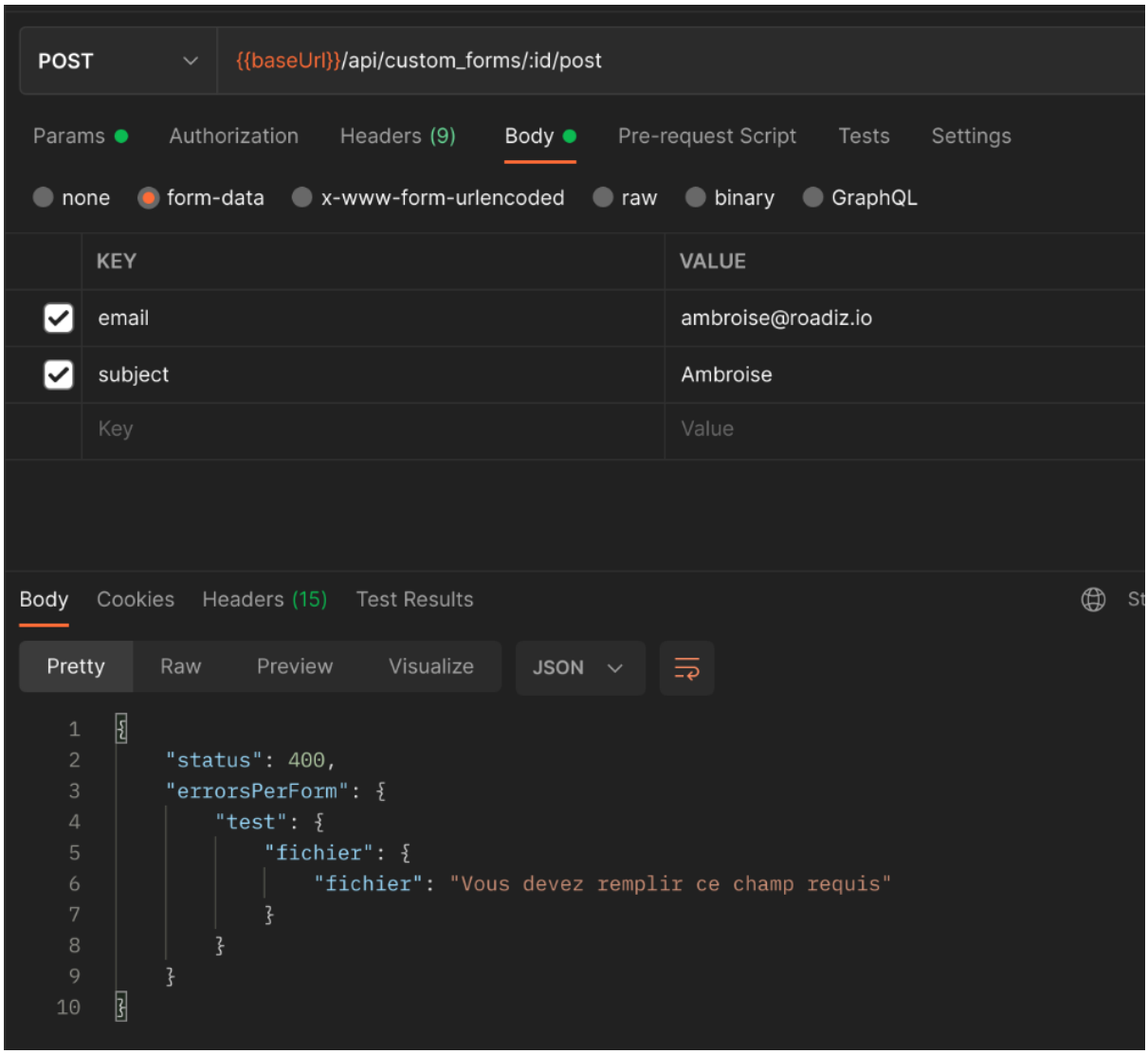
Then you can send your data to the **post** endpoint using *FormData* and respecting field hierarchy:

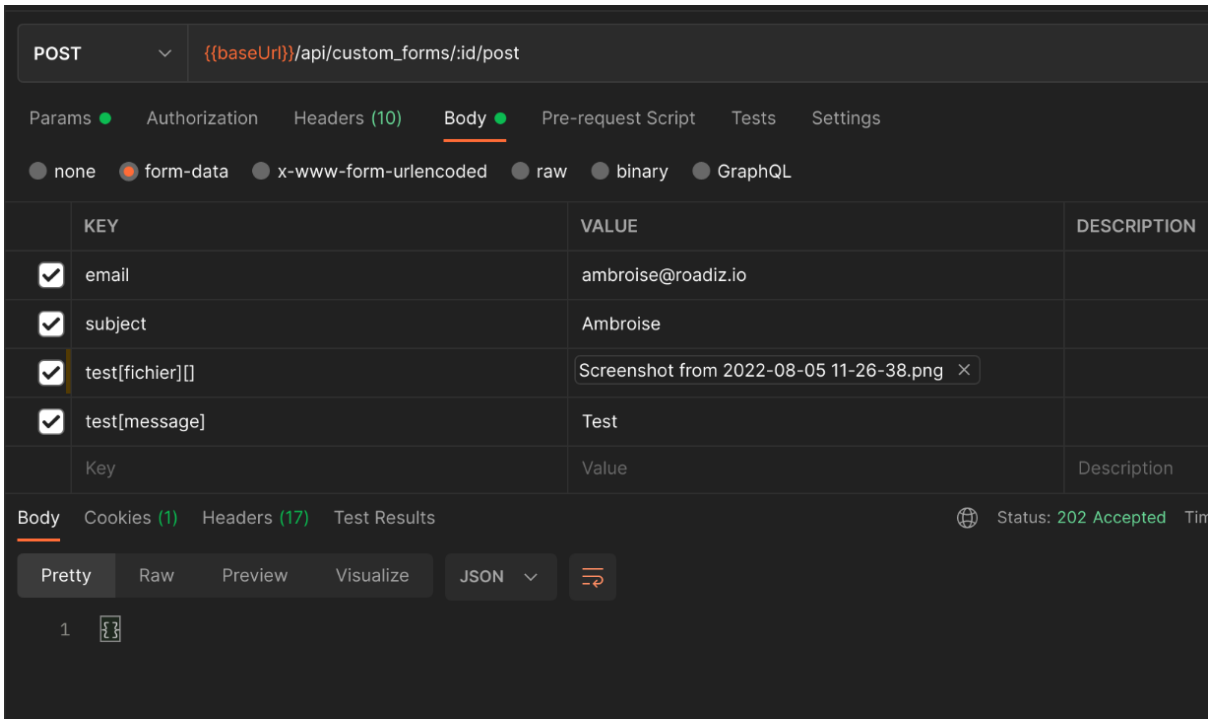
```
POST {{baseUrl}}/api/custom_forms/:id/post
```

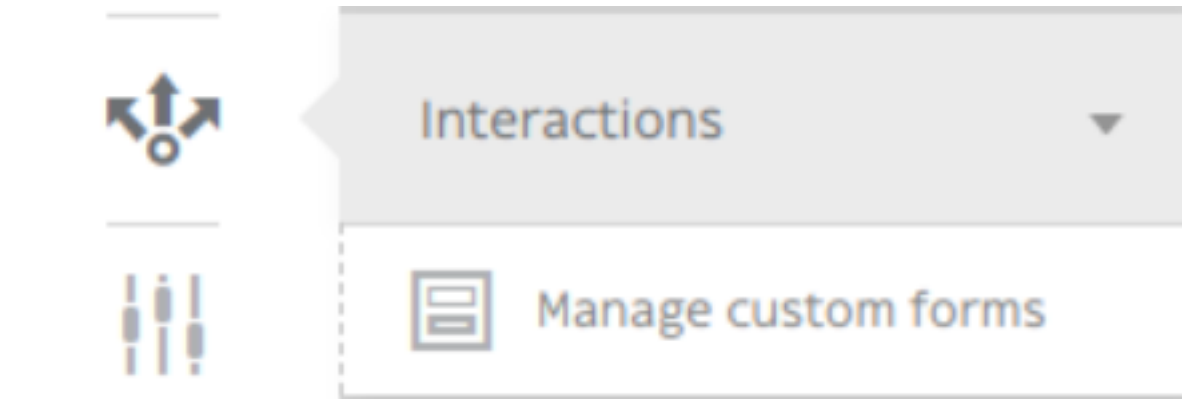
If there are any error, a *JSON* response will give you details fields-by-fields.

If post is successful, APi will respond an empty `202 Accepted` response

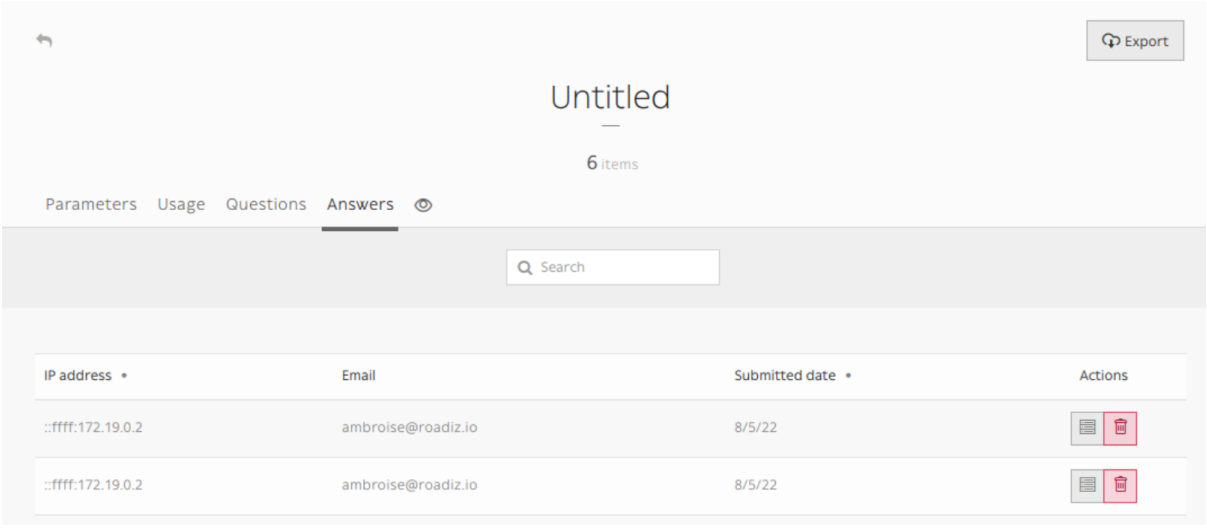
Then you will be able to see all your form submits in Roadiz backoffice :


POST
{{baseUrl}}/api/custom_forms/:id/post
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
KEY
VALUE
email
ambroise@roadiz.io
subject
Ambroise
Key
Value
Body
Cookies
Headers (15)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"status": 400,
"errorsPerForm": {
"test": {
"fichier": {
"fichier": "Vous devez remplir ce champ requis"
}
}
}
}

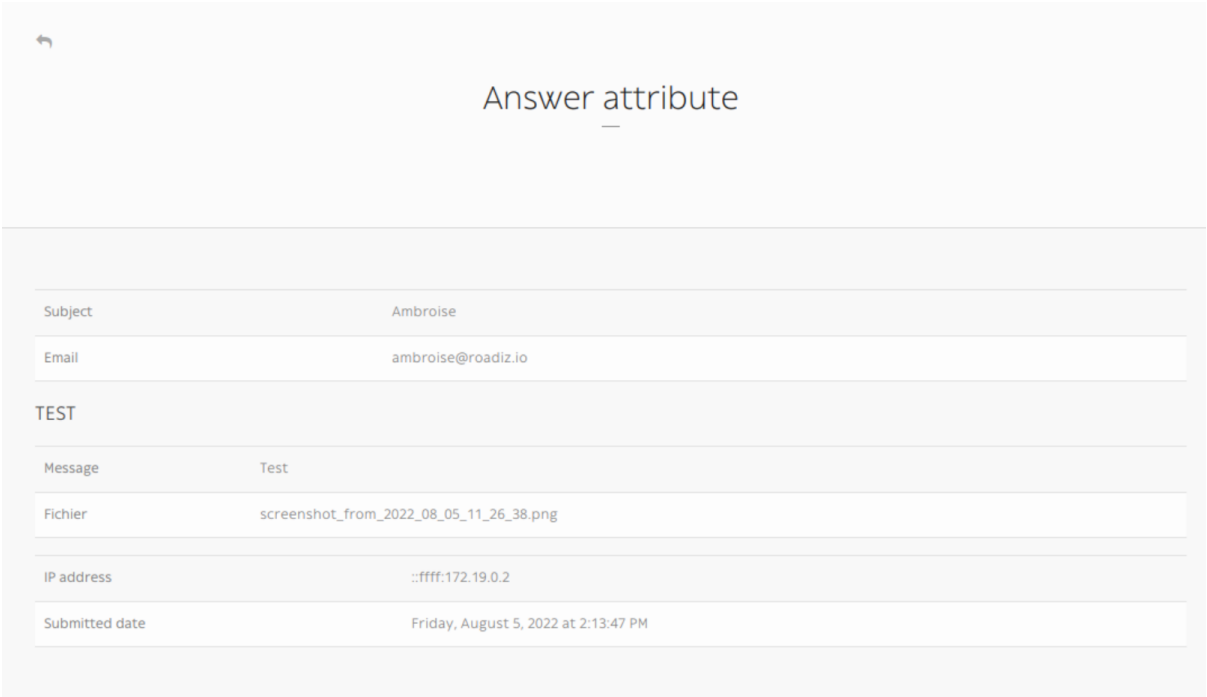

POST
{{baseUrl}}/api/custom_forms/:id/post
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
KEY
VALUE
DESCRIPTION
email
ambroise@roadiz.io
subject
Ambroise
test[fichier][]
Screenshot from 2022-08-05 11-26-38.png
test[message]
Test
Key
Value
Description
Body
Cookies (1)
Headers (17)
Test Results
Status: 202 Accepted
Pretty
Raw
Preview
Visualize
JSON
{}



In Manage custom forms section / Answers



Note: Any file attached to your custom-form answers will be uploaded as private documents.



Adding custom form to your theme

If you want to integrate your custom-forms into your theme, you can use Roadiz `CustomFormHelper` class to generate a standard `FormInterface` and to create a view into your theme templates.

First you must create a dedicated action for your node or your block if you used `{{ nodeSource|render(@AwesomeTheme) }}` Twig filter.

```php
use RZ\Roadiz\CoreBundle\Entity\CustomForm;
use RZ\Roadiz\CoreBundle\Exception\EntityAlreadyExistsException;
use RZ\Roadiz\CoreBundle\Exception\ForceResponseException;
use Symfony\Cmf\Component\Routing\RouteObjectInterface;
use RZ\Roadiz\CoreBundle\CustomForm\CustomFormHelper;
use Symfony\Component\Form\FormError;
use Symfony\Component\HttpFoundation\JsonResponse;

// ...

/*
 * Get your custom form instance from your node-source
 * only if you added a *custom_form reference field*.
 */
$customForms = $this->nodeSource->getCustomFormReference();
if (isset($customForms[0]) && $customForms[0] instanceof CustomForm) {
    /** @var CustomForm $customForm */
    $customForm = $customForms[0];

    /*
     * Verify if custom form is still open
     * for answers
     */
    if ($customForm->isFormStillOpen()) {
        /*
         * CustomFormHelper will generate Symfony form against
         * Roadiz custom form entity.
         * You can add a Google Recaptcha passing following options.
         */
        $helper = $this->customFormHelperFactory->createHelper($customForm);
        $form = $helper->getForm($request, false, true);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            try {
                $answer = $helper->parseAnswerFormData($form, null, $request->
↪getClientIp());

                if ($request->isXmlHttpRequest()) {
                    $response = new JsonResponse([
                        'message' => $this->getTranslator()->trans('form_has_been_
↪successfully_sent')
                    ]);
                } else {
                    $this->publishConfirmMessage(
                        $request,
                        $this->getTranslator()->trans('form_has_been_successfully_
↪sent')
                    );
                    $response = $this->redirect($this->generateUrl(
                        RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
                        [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource->
↪getParent()]
                    ));
                }
```

```
                    /*
                     * If you are in a BlockController use ForceResponseException
                     */
                    throw new ForceResponseException($response);
                    /*
                     * Or directly return redirect response.
                     */
                    //return $response;
                } catch (EntityAlreadyExistsException $e) {
                    $form->addError(new FormError($e->getMessage()));
                }
            }

            $this->assignation['form'] = $form->createView();
        }
    }
```

If you didn't do it yet, create a custom form theme in your `views/` folder:

```
{#
 # AwesomeTheme/Resources/views/form.html.twig
 #}
{% extends "bootstrap_3_layout.html.twig" %}

{% block form_row -%}
    <div class="form-group form-group-{{ form.vars.block_prefixes[1] }} form-
↪group-{{ form.vars.name }}">
        {% if form.vars.block_prefixes[1] != 'separator' %}
            {{- form_label(form) -}}
        {% endif %}
        {{- form_errors(form) -}}
        {#
         # Render field description inside your form
         #}
        {% if form.vars.attr['data-description'] %}
            <div class="form-description">
                {{ form.vars.attr['data-description']|markdown }}
            </div>
        {% endif %}
        {{- form_widget(form) -}}
    </div>
{%- endblock form_row %}

{% block recaptcha_widget -%}
    <input id="my-form-recaptcha" type="hidden" name="{{ form.vars.name }}" />
    <script src="https://www.google.com/recaptcha/api.js?render={{ configs.
↪publicKey }}"></script>
    <script>
        /*
         * Google Recaptcha v3
         * @see https://developers.google.com/recaptcha/docs/v3
         */
        (function() {
            if (!window.grecaptcha) {
                console.warn('Recaptcha is not loaded');
            }
            var form = document.getElementById('my-form');
            form.addEventListener('submit', function (event) {
                event.preventDefault();
                window.grecaptcha.ready(function() {
                    window.grecaptcha.execute('{{ configs.publicKey }}', {action:
↪'submit'}).then(function(token) {
                        var input = document.getElementById('my-form-recaptcha');
```

```
                        if (input) {
                            input.value = token;
                        }
                        form.submit()
                    });
                });
            });
        })();
    </script>
{%- endblock recaptcha_widget %}
```

In your main view, add your form and use your custom form theme:

```
{#
 # AwesomeTheme/Resources/views/form-blocks/customformblock.html.twig
 #}
{% if form %}
    {% form_theme form '@AwesomeTheme/form.html.twig' %}
    {{ form_start(form) }}
    {{ form_widget(form) }}
    <div class="form-group">
        <button class="btn btn-primary" type="submit">{% trans %}send_form{% endtrans %}</button>
    </div>
    {{ form_end(form) }}
{% else %}
    <p class="alert alert-warning">{% trans %}form_is_not_available{% endtrans %}</p>
{% endif %}
```

3.1.8 Contributing

If you want to contribute to Roadiz project by reporting issues or hacking code, let us thank you! You are awesome!

Reporting issues

When you encounter an issue with Roadiz we would love to hear about it. Because thanks to you, we can make the most awesome and stable CMS! If you submit a bug report please include all information available to you, here are some things you can do:

- Try to simplify the things you are doing until getting a minimal set of actions reproducing the problem.
- Do not forget to join a screenshot or a trace of your error.

Coding style

The code you contributed to the project should respect the guidelines defined in PHP *PSR2* standard. If you install the requirements for devs by the command `composer update --dev`, you can use *phpcs* to check your code. You can copy and paste the following command-lines to check easily:

```
php bin/phpcs --report=full --report-file=./report.txt -p ./
```

Or you can use *phpcbf* to automatically fix code style issues.

```
php bin/phpcbf --report=full --report-file=./report.txt -p ./
```

Please take those rules into account, we aim to have a clean codebase. A coherent code-style will contribute to Roadiz stability. Your code will be checked when we will be considering your pull requests.

Static analysis

Then we use `phpstan` as a static code analyzer to check bugs and misuses before they occur:

```
php bin/phpstan analyse -c phpstan.neon
```

3.1.9 Troubleshooting

Empty caches manually for an environment

If you experience errors only on a dedicated environment such as `prod``or ``dev`, it means that cache is not fresh for these environments. As a first try, you should always call `bin/console cache:clear;` (replace *prod* by your environment) in command line.

Problem with entities and Doctrine cache?

After each Roadiz **upgrade** you should always upgrade your node-sources entity classes and upgrade database schema.

```
bin/console generate:nsentities;
bin/console doctrine:schema:update --dump-sql --force;
bin/console cache:clear;
```

CHAPTER 4

Extension documentation

4.1 Extension system

4.1.1 Extending Roadiz

Add back-office entry

At first, create a controller into your theme folder, for example `src/Controller/Admin/AdminController`.

Example:

```
namespace App\Controller\Admin;

use Themes\Rozier\RozierApp;
use Symfony\Component\HttpFoundation\Request;

class AdminController extends RozierApp
{
    public function listAction(
        Request $request
    ) {
        return $this->render(
            'admin/test.html.twig',
            $this->assignation
        );
    }
}
```

If you look at this example you can see the class extends `RozierApp` class. This will enable you to "inject" your code into Rozier Back-stage DOM and Style.

Now let's have a look to your twig template file `templates/admin/test.html.twig`.

```
{% extends '@Rozier/layout.html.twig' %}

{% block customStyles %}
<style>
    /* Custom styles here */
</style>
{% endblock %}
```

```
{% block customScripts %}
<script>
    /* Custom Scripts here */
</script>
{% endblock %}

{% block content %}
<section class="content-global add-test">
    <header class="content-header header-test header-test-edit">
        <h1 class="content-title test-add-title">{% trans %}Test admin{% endtrans
↪%}</h1>
    </header>

    <article class="content content-test">
        <p>This page is created from MyTheme to show you how to extend backoffice␣
↪features.</p>
    </article>
</section>
{% endblock %}
```

The first line is for inheriting from Rozier base template, you can notice that we explicitly choose *@Rozier* namespace.

The two next blocks are made for you to add some CSS or Javascript. For CSS, the block `customStyle` can be use to link an external file with a `<link>` tag, the path must be something like that `{{ asset('static/css/customstyle.css', 'MyTheme') }}`, or add directly some CSS with "<style>" tag. For JS, the block `customScripts` work as is, just link an external JS file or write your `<script>` tag.

Then create your own content, do not hesitate to give a look at Rozier back-stage theme Twig files to use the right DOM structure. For simple features, you wouldn't have to extend JS nor CSS if you follow the same HTML coding style.

Linking things together

Add the route in the theme `config/routes.yaml` file.

In this case the route will be:

```
adminTestPage:
    # Setting your path behind rz-admin will activate Firewall
    path: /rz-admin/test
    defaults:
        _controller: App\Controller\Admin\AdminController::listAction
```

Inject your own entries in back-stage

The last thing to do is to add your new admin entry in the back-office menu.

Go to your `config/packages/roadiz_rozier.yaml` and add your own `entries`:

```
roadiz_rozier:
    entries:
        # ...
        customAdmin:
            name: customAdmin
            route: adminTestPage
            icon: uk-icon-cube
            roles: ~
```

If you want to have a category and sub-entries, just change the path at `null` value and create your `subentries` array as described in the next example:

```
roadiz_rozier:
    entries:
        # ...
        customAdmin:
            name: customAdmin
            route: ~
            icon: uk-icon-cube
            roles: ~
            subentries:
                customAdminPage:
                    name: 'customAdmin page'
                    route: adminTestPage
                    icon: uk-icon-cube
                    roles: ~
```

You can restrict buttons to users with specific roles. Just replace `roles: ~` with `roles: [ 'ROLE_ACCESS_NODES' ]`. You can even create your own roles to take full power of Roadiz extension system.

Warning: Adding roles in `roadiz_rozier.entries` service will only restrict buttons display in Rozier backstage interface. To really protect your controllers from unwanted users add `$this->validateAccessForRole('ROLE_ACCESS_MY_FEATURE');` at the first line of your back-ofice controller's actions. This will kick non-granted users from your custom back-office parts. Give a look at Rozier theme controllers to see how we use it.

4.1.2 Events

Roadiz node system implements several events. So you will be able to create and inject your own event subscribers inside *Roadiz* dispatcher.

To understand how the event dispatcher works, you should read the Symfony documentation at before.

Nodes events

- `RZ\Roadiz\CoreBundle\Event\Node\NodeCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeDeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeUndeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeDuplicatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodePathChangedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeTaggedEvent::class`: This event is triggered for tag and un-tag action.
- `RZ\Roadiz\CoreBundle\Event\Node\NodeVisibilityChangedEvent::class`: This event is triggered each time a node becomes visible or unvisible.
- `RZ\Roadiz\CoreBundle\Event\Node\NodeStatusChangedEvent::class`: This event is triggered each time a node status changes.

Each node event object contains the current `Node` entity. You will get it using `$event->getNode()`.

NodesSources events

`RZ\Roadiz\CoreBundle\Event\NodesSourcesEvents`

- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPreUpdatedEvent::class`: This event is dispatched BEFORE entity manager FLUSHED.

- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesUpdatedEvent::class`: This event is dispatched AFTER entity manager FLUSHED.
- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesDeletedEvent::class`

Each node-source event object contains the current `NodesSources` entity. You will get it using `$event->getNodeSource()`.

- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesIndexingEvent::class`: This event type is dispatched during Solr indexation. Your event will be `\RZ\Roadiz\CoreBundle\Event\FilterSolariumNodeSourceEvent` and it will allow you to alter or improve your Solr index according to your node-source type.

Note: You will find a simple subscriber example in Roadiz back-office theme which is called `Themes\Rozier\Events\SolariumSubscriber`. This subscriber is useful to update or delete your *Solr* index documents against your node-source database.

- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent::class`: This event type is dispatched when the node-router generate a path for your node-source using `{{ path() }}` Twig method or `$this->urlGenerator->generate()` controller method. The default subscriber will generate the complete hierarchical path for any node-source using their identifier (available url-alias or node' name).

Tags events

- `RZ\Roadiz\CoreBundle\Event\Tag\TagCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Tag\TagUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Tag\TagDeletedEvent::class`

Each tag event object contains the current `Tag` entity. You will get it using `$event->getTag()`.

Folders events

- `RZ\Roadiz\CoreBundle\Event\Folder\FolderCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Folder\FolderUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Folder\FolderDeletedEvent::class`

Each folder event object contains the current `Folder` entity. You will get it using `$event->getFolder()`.

Translations events

- `RZ\Roadiz\CoreBundle\Event\Translation\TranslationCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Translation\TranslationUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Translation\TranslationDeletedEvent::class`

Each folder event object contains the current `Translation` entity. You will get it using `$event->getTranslation()`.

UrlAlias events

- `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasDeletedEvent::class`

Each folder event object contains the current `UrlAlias` entity. You will get it using `$event->getUrlAlias()`.

User events

- `RZ\Roadiz\CoreBundle\Event\User\UserCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserDeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserDisabledEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserEnabledEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserPasswordChangedEvent::class`

Each folder event object contains the current `User` entity. You will get it using `$event->getUser()`.

4.1.3 Extending Solr indexation

How to index page blocks contents

If all your text content is written in *block nodes* instead of reachable *pages*, you should index them into your page Solr documents to improve your search engine relevancy.

You can use the `NodesSourcesIndexingEvent::class` event to enhance your node indexing data before it's persisted into *Solr* engine (especially `collection_txt` field):

```php
<?php

declare(strict_types=1);

namespace App\EventSubscriber;

use RZ\Roadiz\CoreBundle\Api\TreeWalker\AutoChildrenNodeSourceWalker;
use RZ\Roadiz\CoreBundle\Entity\NodesSources;
use RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesIndexingEvent;
use RZ\Roadiz\CoreBundle\SearchEngine\SolariumFactoryInterface;
use RZ\TreeWalker\WalkerContextInterface;
use RZ\TreeWalker\WalkerInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;

/**
 * Index sub nodes content into any reachable node-source.
 */
final class NodeSourceIndexingEventSubscriber implements EventSubscriberInterface
{
    private WalkerContextInterface $walkerContext;
    private SolariumFactoryInterface $solariumFactory;
    private int $maxLevel;

    /**
     * @param WalkerContextInterface $walkerContext
     * @param SolariumFactoryInterface $solariumFactory
     * @param int $maxLevel
     */
    public function __construct(
        WalkerContextInterface $walkerContext,
        SolariumFactoryInterface $solariumFactory,
        int $maxLevel = 5
    ) {
        $this->walkerContext = $walkerContext;
        $this->solariumFactory = $solariumFactory;
        $this->maxLevel = $maxLevel;
    }

    /**
     * @inheritDoc
```

```php
     */
    public static function getSubscribedEvents(): array
    {
        return [
            NodesSourcesIndexingEvent::class => ['onIndexing'],
        ];
    }

    public function onIndexing(NodesSourcesIndexingEvent $event): void
    {
        $nodeSource = $event->getNodeSource();

        if (null !== $nodeSource->getNode() && $nodeSource->isReachable() && !
↪$event->isSubResource()) {
            $assoc = $event->getAssociations();

            $blockWalker = AutoChildrenNodeSourceWalker::build(
                $nodeSource,
                $this->walkerContext,
                $this->maxLevel
            );

            // Need a locale field
            $locale = $nodeSource->getTranslation()->getLocale();
            $lang = \Locale::getPrimaryLanguage($locale) ?? 'fr';

            foreach ($blockWalker->getChildren() as $subWalker) {
                $this->walkAndIndex($subWalker, $assoc, $lang);
            }

            $event->setAssociations($assoc);
        }
    }

    /**
     * @param WalkerInterface $walker
     * @param array $assoc
     * @param string $locale
     * @throws \Exception
     */
    protected function walkAndIndex(WalkerInterface $walker, array &$assoc, string
↪$locale): void
    {
        $item = $walker->getItem();
        if ($item instanceof NodesSources) {
            $solarium = $this->solariumFactory->createWithNodesSources($walker->
↪getItem());
            // Fetch all fields array association AS sub-resources (i.e. do not␣
↪index their title)
            $childAssoc = $solarium->getFieldsAssoc(true);
            $assoc['collection_txt'] = array_filter(array_merge(
                $assoc['collection_txt'],
                $childAssoc['collection_txt']
            ));
            if (!empty($childAssoc['collection_txt_' . $locale])) {
                $assoc['collection_txt_' . $locale] .= PHP_EOL . $childAssoc[
↪'collection_txt_' . $locale];
            }
        }
        if ($walker->count() > 0) {
            foreach ($walker->getChildren() as $subWalker) {
                $this->walkAndIndex($subWalker, $assoc, $locale);
            }
```

```
            }
        }
}
```

Index

A

attribute, 65
attribute_documents, 65

C

color, 65
custom_form, 65

D

document, 65
document_display, 65
document_display_sources, 65
document_folders, 65
document_folders_all, 65
document_private, 65
document_thumbnails, 65

F

folder, 65

I

id, 64

N

node, 65
node_attributes, 65
node_children, 65
node_listing, 65
node_type, 65
nodes_sources, 65
nodes_sources_base, 65
nodes_sources_default, 65
nodes_sources_documents, 65
nodes_sources_‘‘group‘‘, 65

P

position, 65

S

setting, 65
setting_group, 65

T

tag, 65
tag_base, 65
tag_children, 65
tag_parent, 65
timestamps, 64
translation, 65
translation_base, 65

U

user, 65
user_group, 65
user_identifier, 66
user_personal, 66
user_role, 65